UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES

ANALISIS EMPIRICO DE LOS COSTOS DE COMUNICACIÓN EN UN SISTEMA DISTRIBUIDO, PARA ALGORITMOS DE DISTRIBUCIÓN DE CARGA EN BASE DE DATOS ESPACIO TEMPORALES.

AUTOR:

Esteban Adolfo Acevedo Sánchez.

PROFESOR GUÍA:

Sr. Claudio Orlando Gutiérrez Soto.

CARRERA:

Ingeniería Civil Informática.

Concepción, 2016.

Índice.

Capítulo I 8

1. Generalidades 9

1.1. Origen del Tema 9

1.2. Justificación 9

1.3. Objetivos 11

1.3.1. Objetivo General 11

1.3.2. Objetivos Específicos 11

1.4. Aporte 12

1.5. Límites 12

1.6. Resumen 12

Capítulo II 13

2. Bases de Datos Espacio Temporales 14

2.1. Métodos de Acceso Espacio-Temporal 15

2.1.1. R-Tree 16

2.1.2. 3D R-Tree 17

2.1.3. Mvb-Tree 18

2.1.4. Hr-Tree. 18

2.1.5. Mvr-Tree 19

2.1.6. Mv3r-Tree 20

2.2. Resumen Capítulo 20

Capítulo III 21

3. Procesamiento Paralelo y Diseño del Algoritmo 22

3.1. Diseño del Algoritmo Paralelo En Línea. 26

3.1.1. Estrategia de División del Dominio 26

3.1.2. Estrategia Local de Paralelización 27

3.1.3. Estrategia de Distribución Circular 28

3.1.4. Estrategia de un Algoritmo En Línea 29

3.1.5. Descripción del Algoritmo En Línea 29

3.1.6. Objetos 31
3.1.7. Consultas… 32
3.1.8. Creación Archivo Set 33
3.1.9. Leer Objetos 34
3.1.10. Obtención de objetos móviles, estáticos y estadísticas de los objetos 35
3.1.11. Distribución de Objetos 36
3.1.12. Creación de Consultas 38
3.1.13. Código Mvr-Paralelo 38
3.1.14. Detalles Generales Implementación Paralela 39
3.1.15. Detalles Codificación 40
3.2. Resumen Capítulo 41
Capítulo IV 42
4. Diseño de Escenario Experimental 43
4.1. Distribuciones 43
4.1.1. Distribución Zipf 43
4.1.2. Distribución Gamma 43
4.1.3. Distribución Exponencial 43
4.2. Diseño de Escenario Experimental 44
4.3. Resultados 44
4.3.1. Distribución Exponencial 45
4.3.1.1. Experimento 1.1 50 TS/50 (700) 45
4.3.1.2. Experimento 2.1 50 TI/50 TS (700) 45
4.3.1.3. Experimento 3.1 1 TI/1 TS (700) 46
4.3.1.4. Experimento 1.1 50 Ts/50 Ti (500) 46
4.3.1.5. Experimento 2.1 50 Ti/50 Ts (500) 47
4.3.1.6. Experimento 3.1 1 Ti/1 Ts (500) 47
4.3.1.7. Experimento 1.1 50 Ts/50 Ti (300) 48
4.3.1.8. Experimento 2.1 50 Ti/50 Ts (300) 48
4.3.1.9. Experimento 3.1 1 Ti/1 Ts (300) 49

4.3.1.10. Experimento 1.1 50 TS/50 (700) 49
4.3.1.11. Experimento 2.1 50 TI/50 TS (700) 50
4.3.1.12. Experimento 3.1 1 TI/1 TS (700) 50
4.3.1.13. Experimento 1.1 50 Ts/50 Ti (500) 51
4.3.1.14. Experimento 2.1 50 Ti/50 Ts (500) 51
4.3.1.15. Experimento 3.1 1 Ti/1 Ts (500) 52
4.3.1.16. Experimento 1.1 50 Ts/50 Ti (300) 52
4.3.1.17. Experimento 2.1 50 Ti/50 Ts (300) 53
4.3.1.18. Experimento 3.1 1 Ti/1 Ts (300) 53
4.3.2. Distribución Gamma 54
4.3.2.1. Experimento 1.2 50 TS/50 (700) 54
4.3.2.2. Experimento 2.2 50 TI/50 TS (700) 54
4.3.2.3. Experimento 3.2 1 TI/1 TS (700) 55
4.3.2.4. Experimento 1.2 50 Ts/50 Ti (500) 55
4.3.2.5. Experimento 2.2 50 Ti/50 Ts (500) 56
4.3.2.6. Experimento 3.2 1 Ti/1 Ts (500) 56
4.3.2.7. Experimento 1.2 50 Ts/50 Ti (300) 57
4.3.2.8. Experimento 2.2 50 Ti/50 Ts (300) 57
4.3.2.9. Experimento 3.2 1 Ti/1 Ts (300) 58
4.3.3. Distribución Zipf 58
4.3.3.1. Experimento 1.3 50 TS/50 (700) 58
4.3.3.2. Experimento 2.3 50 TI/50 TS (700) 59
4.3.3.3. Experimento 3.3 1 TI/1 TS (700) 59

4.3.3.4. Experimento 1.3 50 Ts/50 Ti (500) 60
4.3.3.5. Experimento 2.3 50 Ti/50 Ts (500) 60
4.3.3.6. Experimento 3.3 1 Ti/1 Ts (500) 61
4.3.3.7. Experimento 1.3 50 Ts/50 Ti (300) 61
4.3.3.8. Experimento 2.3 50 Ti/50 Ts (300) 62
4.3.3.9. Experimento 3.3 1 Ti/1 Ts (300) 62
4.3.3.10. Experimento 1.3 50 TS/50 (700) 63
4.3.3.11. Experimento 2.3 50 TI/50 TS (700) 63
4.3.3.12. Experimento 3.3 1 TI/1 TS (700) 64
4.3.3.13. Experimento 1.3 50 Ts/50 Ti (500) 64
4.3.3.14. Experimento 2.3 50 Ti/50 Ts (500) 65
4.3.3.15. Experimento 3.3 1 Ti/1 Ts (500) 65
4.3.3.16. Experimento 1.3 50 Ts/50 Ti (300) 66
4.3.3.17. Experimento 2.3 50 Ti/50 Ts (300) 66
4.3.3.18. Experimento 3.3 1 Ti/1 Ts (300) 67
Capítulo V 68
5. Conclusiones…. 69
Referencias..... 70
Anexo……. 74

Ilustración 1: Evolución Índices Espacio Temporales. 15
Ilustración 2: Árbol R-Tree. 16
Ilustración 3: 3D R-TREE (Theodoridis et al., 1996) 17
Ilustración 4: Mvb-Tree. 18

Ilustración 5: Hr-Tree. 19
Ilustración 6: Ejemplo de distribución de datos para N=1000 y P=4. 27
Ilustración 7: Asignación de datos en Estrategia de Distribución Circular para P=4 y N=1000. 28
Ilustración 8: Distribución objetos móviles. 30
Ilustración 9: Distribución objetos estáticos. 30
Ilustración 10: Seudo Código Algoritmo 31
Ilustración 11: Ilustración MBR. 32
Ilustración 12: Archivo Consultas.txt 33
Ilustración 13: Lista Ligada por Punteros 34
Ilustración 14: Seudo Código para Crear Set. 34
Ilustración 15: Seudo Código para Leer Objetos. 34
Ilustración 16: Seudo Código para Generar Estadísticas. 35
Ilustración 17: Lista Ligada por Punteros de los Archivo Móviles, Data set y archivo Estáticos. 36
Ilustración 18: Distribución Lista Ligada por Punteros del Archivo Móviles, 4 Archivos/4 Procesadores. 37
Ilustración 19: Seudo Código para Distribuir los Objetos. 37
Ilustración 20: Seudo Código para Generar Consultas. 38
Ilustración 21: Frecuencia de aparición de los números a generar 43
Ilustración 22: experimento 1.1, exponencial = 1, revisión de consulta = 700 45
Ilustración 23: experimento 2.1, exponencial = 1, revisión de consulta = 700 45
Ilustración 24: experimento 3.1, exponencial = 1, revisión de consulta = 700 46
Ilustración 25: experimento 1.1, exponencial = 1, revisión de consulta = 500 46
Ilustración 26: experimento 2.1, exponencial = 1, revisión de consulta = 500 47
Ilustración 27: experimento 3.1, exponencial = 1, revisión de consulta = 500 47
Ilustración 28: experimento 1.1, exponencial = 1, revisión de consulta = 300 48
Ilustración 29: experimento 2.1, exponencial = 1, revisión de consulta = 300 48
Ilustración 30: experimento 3.1, exponencial = 1, revisión de consulta = 300 49
Ilustración 31: experimento 1.1, exponencial = 0.5, revisión de consulta = 700 49
Ilustración 32: experimento 2.1, exponencial = 0.5, revisión de consulta = 700 50
Ilustración 33: experimento 3.1, exponencial = 0.5, revisión de consulta = 700 50
Ilustración 34: experimento 1.1, exponencial = 0.5, revisión de consulta = 500 51
Ilustración 35: experimento 2.1, exponencial = 0.5, revisión de consulta = 500 51
Ilustración 36: experimento 3.1, exponencial = 0.5, revisión de consulta = 500 52
Ilustración 37: experimento 1.1, exponencial = 0.5, revisión de consulta = 300 52
Ilustración 38: experimento 2.1, exponencial = 0.5, revisión de consulta = 300 53
Ilustración 39: experimento 3.1, exponencial = 0.5, revisión de consulta = 300 53

Ilustración 40: experimento 1.2, gamma = 1, revisión de consulta = 700 54
Ilustración 41: experimento 2.2, gamma = 1, revisión de consulta = 700 54
Ilustración 42: experimento 3.2, gamma = 1, revisión de consulta = 700 55
Ilustración 43: experimento 1.2, gamma = 1, revisión de consulta = 500 55
Ilustración 44: experimento 2.2, gamma = 1, revisión de consulta = 500 56
Ilustración 45: experimento 3.2, gamma = 1, revisión de consulta = 500 56
Ilustración 46: experimento 1.2, gamma = 1, revisión de consulta = 300 57
Ilustración 47: experimento 2.2, gamma = 1, revisión de consulta = 300 57
Ilustración 48: experimento 3.2, gamma = 1, revisión de consulta = 300 58
Ilustración 49: experimento 1.3, zipf = 1, revisión de consulta = 700 58
Ilustración 50: experimento 2.3, zipf = 1, revisión de consulta = 700 59
Ilustración 51: experimento 3.3, zipf = 1, revisión de consulta = 700 59
Ilustración 52: experimento 1.3, zipf = 1, revisión de consulta = 500 60
Ilustración 53: experimento 2.3, zipf = 1, revisión de consulta = 500 60
Ilustración 54: experimento 3.3, zipf = 1, revisión de consulta = 500 61
Ilustración 55: experimento 1.3, zipf = 1, revisión de consulta = 300 61
Ilustración 56: experimento 2.3, zipf = 1, revisión de consulta = 300 62
Ilustración 57: experimento 3.3, zipf = 1, revisión de consulta = 300 62
Ilustración 58: experimento 1.3, zipf = 2, revisión de consulta = 700 63
Ilustración 59: experimento 2.3, zipf = 2, revisión de consulta = 700 63
Ilustración 60: experimento 3.3, zipf = 2, revisión de consulta = 700 64
Ilustración 61: experimento 1.3, zipf = 2, revisión de consulta = 500 64
Ilustración 62: experimento 2.3, zipf = 2, revisión de consulta = 500 65
Ilustración 63: experimento 3.3, zipf = 2, revisión de consulta = 500 65
Ilustración 64: experimento 1.3, zipf = 2, revisión de consulta = 300 66
Ilustración 65: experimento 2.3, zipf = 2, revisión de consulta = 300 66
Ilustración 66: experimento 3.3, zipf = 2, revisión de consulta = 300 67

Capítulo I

1. Generalidades

1.1. Origen del Tema

El presente trabajo se enmarca dentro del grupo de investigación de computación obicua, presentado por el académico Claudio Gutiérrez Soto de la Universidad del Bío-Bío. De esta forma, surge la necesidad de contar con alumnos memoristas que puedan trabajar activamente en la investigación, con la finalidad de poder aportar con trabajo y conocimiento.

Es por esta razón que se propone el análisis, diseño e implementación de diversas estrategias de comunicación para obtener los costos comunicación en un sistema distribuido, utilizando un algoritmo en línea para el balance de carga en una Base de Datos Espacio-Temporales.

1.2. Justificación

Las Bases de Datos Espacio-temporales están compuestas de objetos espaciales que cambian su posición y/o forma a lo largo del tiempo. El objetivo es modelar y representar la naturaleza dinámica de las aplicaciones en el mundo real. (Theodoridis et al., 1998).

En el ámbito de estas Bases de Datos, existen dos consultas típicas a responder, las consultas time-slice, que recuperan todos los objetos que se encuentran en un espacio y en un instante particular de tiempo. Por otro lado, la consulta del tipo time-interval extienden la consulta anterior a un intervalo de tiempo. (Chen et al., 2003); (Theodoris et al., 1998).

Un aspecto importante a considerar es la tendencia de las Bases de Datos Espaciales a ser muy grandes. Dicha capacidad de procesamiento se puede obtener a través de varios procesadores trabajando de manera simultánea sobre una misma tarea. A esta capacidad de trabajo se le conoce como paralelismo; y a través de ésta es posible obtener mejores rendimientos que utilizando un solo procesador.

Los Sistemas de Información Geográficos, así como los recientes avances en telecomunicaciones (por ejemplo, los GPS, y redes de celular por nombrar algunos) han facilitado el crecimiento de grandes colecciones de Base de Datos Espacio-Temporales. El volumen de tales datos, así como la necesidad de actualización de consultas e inserción de datos, hace extremadamente difícil un desempeño razonable de estas Base de Datos. Es por ello que para poder obtener un mejor el rendimiento de dichas Base de Datos se hace necesario contar con una alta capacidad de procesamiento. Sin embargo, para lograr una paralelización eficiente es necesario hacer una buena distribución de la carga inicial de trabajo. A dicha distribución inicial se le conoce con el nombre de “División del Dominio”.

Una buena división del dominio asegura que todos los procesadores tendrán una carga inicial similar, lo que se traduce en un buen punto de partida para la eficiencia general del sistema. En ambientes paralelos con memoria y procesadores distribuidos, la comunicación entre los procesadores juega un rol fundamental en el desempeño de sistema. Un exceso de la comunicación trae consigo un desempeño pobre del sistema. Dicha comunicación es utilizada para realizar un balance de carga entre los procesadores durante todo el tiempo que dure el procesamiento de la tarea. Es por ello que, para obtener un rendimiento adecuado en ambientes distribuidos es necesario realizar una adecuada División del Dominio, hacer un uso eficiente de la Comunicación, así como también de la Distribución de Carga entre los procesadores.

El objetivo principal de este proyecto, es la implementación eficiente de estrategias de comunicación en un ambiente distribuido que permitan obtener una distribución de carga adecuada en los procesadores utilizando un algoritmo en línea para el procesamiento de consultas (time-slice y time-interval) con un rendimiento aceptable. De esta forma, se podrá contar con una implementación eficiente que permita obtener respuestas a consultas del tipo time-slice y time-interval en un ambiente distribuido.

Es importante mencionar que se utilizarán algunas aproximaciones estudiadas previamente en tres proyectos de título. En dos proyectos de título se estudia la eficiencia de respuestas a consultas del tipo time-interval y time-slice, usando el índice HR-Tree, y el índice MVR-Tree (además de presentar una primera aproximación de un algoritmo en línea) en el tercer proyecto, se analizó y contrastó el algoritmo en línea basado en cache, versus otro algoritmo basado en colas de prioridad, estos tres proyectos fueron realizados en memoria compartida.

1.3. Objetivos

1.3.1. Objetivo General

Analizar, diseñar e implementar diversas estrategias de comunicación para obtener los costos comunicación en un sistema distribuido, utilizando un algoritmo en línea para el balance de carga en una Base de Datos Espacio-Temporales. Para ello se utilizarán los enfoques (algoritmos) de tres proyectos de título anteriores, con el propósito analizar, diseñar, implementar y obtener resultados experimentales de los costos de comunicación de dichos algoritmos en un ambiente distribuido.

Es importante mencionar que el aporte de este proyecto de título se centra en:

- Obtener los costos de comunicación en un ambiente distribuido, las cuales no fueron abordadas en los proyectos de título mencionados anteriormente.

- Obtener un solo algoritmo de balance de carga en línea, que considere lo mejor de los algoritmos propuestos en los proyectos de título anteriormente mencionados.

- Extender los ambientes experimentales.

1.3.1. Objetivos Específicos

- Estudiar los algoritmos propuestos en los tres proyectos de título anteriores.
- Analizar y diseñar diversas estrategias de comunicación en un ambiente distribuido para los algoritmos propuestos en los proyectos de título anteriores.
- Analizar y determinar distintos escenarios de experimentación acorde a la distribución de los objetos en movimiento (esto es sobre el espacio), así como también considerando los tipos de consultas más frecuentes (en este proyecto de título consideraremos sólo 2, time-slice y time-interval).
- Implementar y obtener resultados empíricos de las estrategias de comunicación en un ambiente distribuido para los proyectos de título mencionados anteriormente**.**
- Extender los algoritmos de los proyectos de título anteriores, en un sólo algoritmo (un nuevo algoritmo o un algoritmo extendido de los previos) que provea los mejores resultados de comunicación para el balance de carga de consultas time-slice y time-interval en una base de datos espacio-temporal.
- Obtener los resultados empíricos del algoritmo extendido en un ambiente distribuido.

1.4. Aporte

El estudio de las Bases de Datos Espacio-Temporales, especialmente relacionados a métodos de acceso, sus estructuras de datos y algoritmos, es materia bastante reciente de investigación. De hecho, muchas de las estrategias de indexación y algoritmos están aún en experimentación, dejando mucho camino por avanzar y muchas áreas en las que es posible realizar un aporte tangible y significativo. Si bien existen propuestas para el control de concurrencia en Data Warehouse y servidores MOLAP (Multidimensional On Line Analytical Processing) sobre los métodos Mvr-Tree y RTree, no se han creado estrategias de paralelización eficientes de estos métodos que permitan optimizar las consultas en bases de datos espacio-temporales.

Además, debemos tener en cuenta que la eficiencia en las respuestas de una Base de Datos toma importancia al momento de crear los algoritmos y estructuras de datos que las soporten, tema que cobra mayor dificultad en Bases de Datos Espacio-Temporales, en donde la cantidad de datos y actualizaciones necesarias son mayores, como se hizo notar anteriormente.

Por lo expuesto anteriormente, creo que mi trabajo realmente será un aporte al estudio de esta área, que sentando bases sólidas incrementará el desarrollo de aplicaciones basadas en objetos espacio temporales que tanto auge ha tenido en los últimos años.

1.5. Límites

Los algoritmos de los métodos de acceso mencionados sólo serán estudiados y no se les realizarán modificaciones a ellos o sus estructuras de datos asociadas para cumplir con las mejoras de eficiencia.

1.6. Resumen

- **Capítulo 2**: En este capítulo se expondrán todos los tópicos teóricos de las bases de datos espacio temporales, describiendo los métodos de acceso espacio temporal y la base de datos espacio temporales.
- **Capítulo 3**: En este capítulo se expondrán todos los tópicos referentes al procesamiento paralelo, además de explicar detalladamente el algoritmo diseñado.
- **Capítulo 4**: En este capítulo se explicará la construcción del escenario experimental.
- **Capítulo 5**: En este capítulo se describirán las conclusiones de la investigación.

Capítulo II

2. Bases de Datos Espacio Temporales

Las Bases de Datos Espacio-Temporales surgen de la necesidad de capturar la evolución de objetos espaciales a lo largo del tiempo y reúnen la funcionalidad de las Bases de Datos Espaciales y las Bases de Datos Temporales.

Por un lado, el aspecto espacial permite manejar información de objetos localizados en algún lugar determinado, siendo posible almacenar: puntos, líneas o áreas. Los puntos representan objetos en donde sólo conocer su posición es importante, por ejemplo, las distintas ciudades de Chile. Las poli líneas son usadas para modelar objetos para los que es relevante conocer el largo, como ríos, calles, carreteras, entre otros. Finalmente, las áreas o regiones poli griales son objetos cuya posición, tamaño y forma son importantes; tal es el caso de regiones, países, parcelas, entre otros.

Por otro lado, el aspecto temporal permite almacenar los cambios de posición y/o forma de estos objetos a lo largo del tiempo.

Las Bases Datos Espacio-Temporales poseen características que las hacen únicas, las cuales se exponen en (Ahn *et al.*, 2001) y son:

- Tienen una estructura compleja.
- Poseen gran dinamismo.
- Las bases de datos tienden a ser grandes.
- No existe un álgebra espacial estándar.

Dentro de las Bases de Datos Espacio-Temporales encontramos las de tipo Histórica y las de tipo Predictiva (Chen *et al.*, 2003). Las Históricas almacenan, para cada objeto contenido, los cambios que ha tenido su extensión espacial a lo largo del tiempo. Por su parte las Predictivas almacenan, para cada objeto, su localización más reciente y la función que describe su movimiento actual.

Las Bases de Datos Espacio-Temporales tienen como objetivo responder consultas que involucren en sus predicados un instante de tiempo determinado o un intervalo dado.

En Chen *et al.*, 2003 y Theodoris *et al.* 1998 se especifican los principales tipos de consulta, entre las cuales tenemos: consultas Time-Slice, consultas Time-Interval, consultas de Join, consultas de vecino más cercano, consultas de evento. Para los fines de esta investigación sólo toman relevancia las consultas Espacio-Temporales de tipo Time-Slice y Time-Interval.

Las consultas de tipo Time-Slice son aquellas que tiene como finalidad encontrar los objetos que se encuentren en un espacio e instante de tiempo dado, y las consultas de tipo Time-Interval tiene como fin encontrar los objetos que se encuentren en un espacio determinado en un intervalo de tiempo dado.

2.1. Métodos de Acceso Espacio-Temporal

Dentro de los métodos de acceso espacio temporal que permiten la recuperación histórica de los objetos existe la siguiente clasificación (Gutiérrez et al., 2006):

- Métodos que tratan el tiempo como una dimensión.
- Métodos que incorporan el tiempo dentro de los nodos de la estructura, pero sin considerarlo como otra dimensión.
- Métodos que usan overlapping (superposición) con la finalidad de representar el estado de la Base de Datos en diferentes instantes de tiempo.
- Métodos basados en la multi versión de la estructura.

La evolución de los índices se mostrará en la figura **1**:

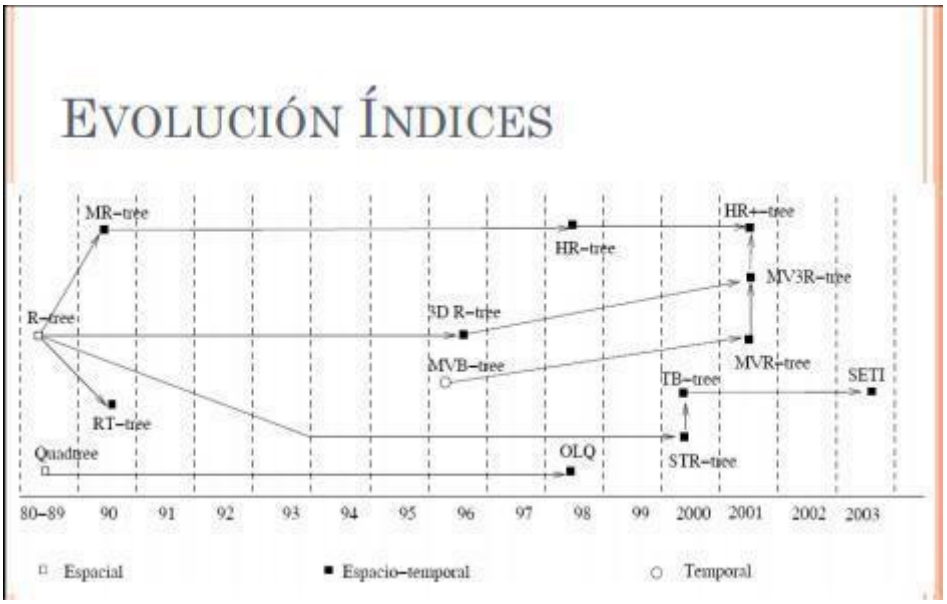


Figura 1: Evolución Índices Espacio Temporales.

A continuación, se representan algunos de los métodos de acceso Espacio-Temporales estudiados en el Proyecto de Título.

2.1.1. R-Tree

El R-Tree es una estructura de datos de tipo árbol similar a los árboles-B, con la diferencia de que se utilizan para métodos de acceso espacial, es decir para indexar información multidimensional, fue adoptado por Oracle y PostgreSQL.

En el índice se almacenan objetos que corresponden a una aproximación geométrica de los objetos reales, usualmente se usa un rectángulo mínimo denominado MBR (Mínimum Bounding Rectangle) que es capaz de contener completamente la geometría o atributos espaciales de los objetos. A continuación, en la figura **2** se mostrará un árbol R-TREE.

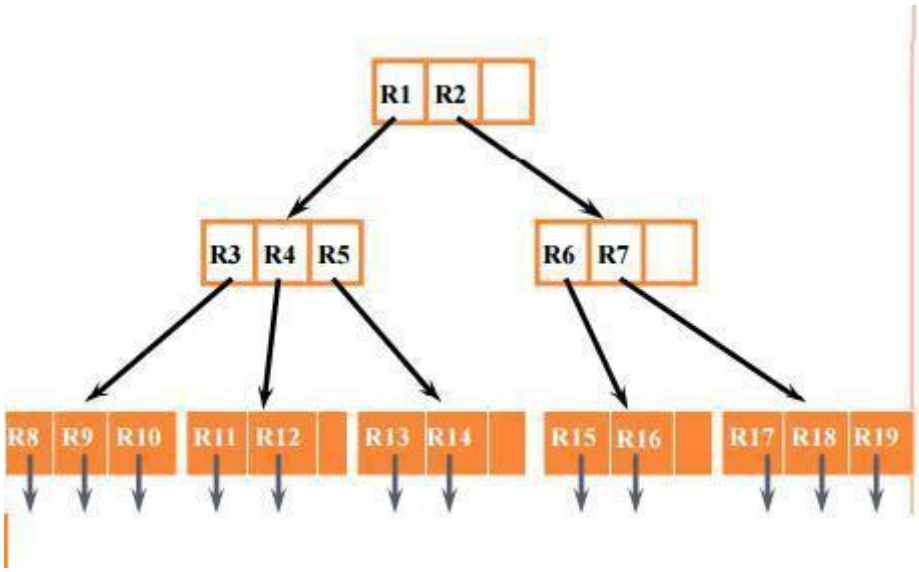


Figura 2: Árbol R-Tree.

- Cada nodo corresponde a una página o bloque de disco.
- Los nodos hojas de un R-TREE contienen entradas de la forma (MBR, oid) donde oid es el identificador del objeto espacial en la BD y MBR es un rectángulo multidimensional que contiene espacialmente al objeto.
- Los nodos intermedios contienen entradas de la forma (MBR, ref) donde ref es la dirección del correspondiente nodo hijo en el R-TREE y MBR es el rectángulo mínimo que contiene a todos los rectángulos definidos en las entradas el nodo hijo.

2.1.2. 3D R-Tree

El 3D R-Tree (Theodoridis et al., 1996) es uno de los métodos que ven el tiempo como otra dimensión y lo integran a la construcción del árbol junto con las otras dimensiones. La ilustración **3** muestra (a) un conjunto de objetos espaciales con un período de vida específico para cada uno y (b) el 3D R-Tree correspondiente.

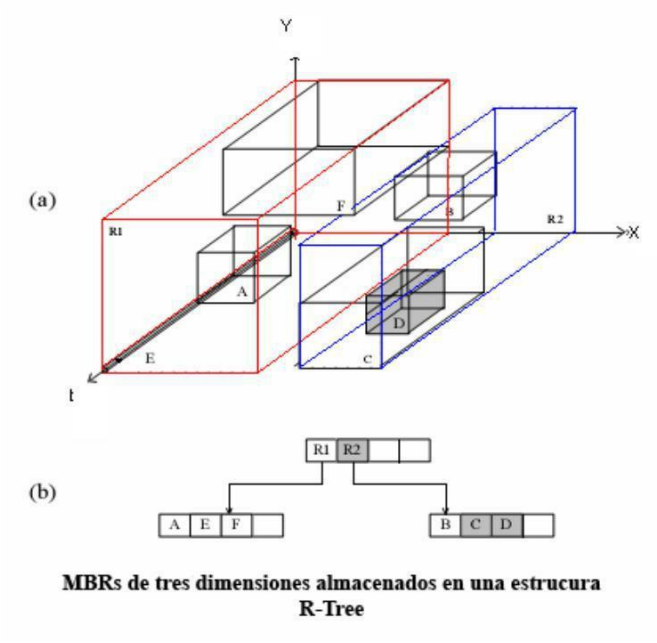


Ilustración 3: 3D R-TREE (Theodoridis et al., 1996)

El movimiento de los objetos en dos dimensiones puede ser modelado como rectángulos distintos dentro del espacio de tres dimensiones. La proyección temporal indica el período en donde el correspondiente objeto permanece estático, mientras que las proyecciones espaciales del rectángulo corresponden a la posición y extensiones de los objetos durante el período. Cada vez que un objeto se mueve a otra posición, un nuevo rectángulo es creado para representar su nuevo período estático, posición y extensiones.

2.1.3. Mvb-Tree

El Mvb-Tree (Becker *et al*., 1996) es una estructura parcialmente persistente en el sentido de que las inserciones y eliminaciones sólo pueden ocurrir en el tiempo actual. Las eliminaciones son lógicas: el registro actual no es removido físicamente de la base de datos. La ilustración 4 muestra un sencillo ejemplo, donde cada entrada tiene la forma <clave, T_{start}, T_{end}, *puntero*>. Para los nodos hoja el puntero apunta al actual registro con el valor correspondiente a clave, T_{start} denota el tiempo en que el registro fue insertado en la Base de Datos, y T_{end} el tiempo en que fue borrado. Para los nodos intermedio, el puntero apunta al nodo de nivel siguiente, mientras T_{start} y T_{end} son los valores máximo y mínimo respectivamente en este nodo. Una entrada se dice activa en un instante de tiempo *t* si T_{start} si $t_{start} \leq t \leq t_{end}$, e inactiva en otro caso (notar que el lapso de actividad de una entrada no incluye t_{end}). El valor de t_{end} para las entradas actualmente activas es "*", que denota la palabra reservada *"NOWTIME"*. Si se inserta una nueva entrada en el instante de tiempo *t*, t_{start} toma el valor de *t* y t_{end} toma el valor "*". Por otro lado, si una entrada es borrada lógicamente, t_{end} cambia de "*" a *t*.

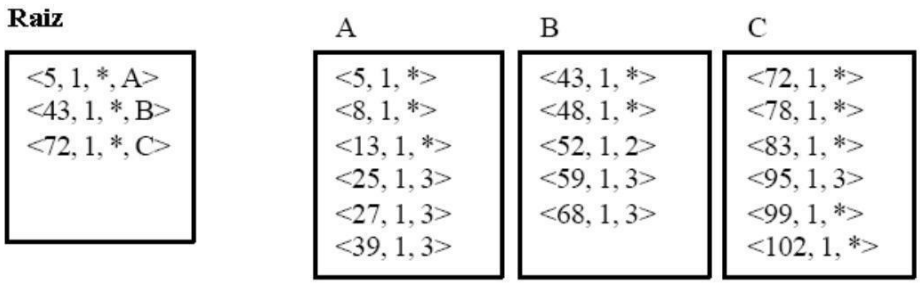


Ilustración 4: Mvb-Tree.

Puede haber múltiples raíces en un Mvb-Tree y cada raíz tiene un intervalo de jurisdicción que es el mínimo lapsus de actividad de todas las entradas en la raíz.

2.1.4. Hr-Tree.

El Historical R-Tree o Hr-Tree (Nascimento y Silva, 1998) mantiene un R-Tree para cada marca de tiempo o Time Stamp, pero las ramas comunes de árboles consecutivos se almacenan solamente una vez con el fin de ahorrar espacio. La ilustración **5**, muestra parte de un HR-TREE para los instantes de tiempo T0 (a) y T1 (b) donde los nodos B y C son compartidos por ambos árboles (c), ya que su contenido no ha cambiado durante estos Time

Stamps. En T1 el objeto 3 cambia su posición (b), así esta entrada debe ser borrada del árbol para T1, mientras se inserta su nueva posición. Luego, para almacenar la nueva entrada se debe crear un nuevo nodo hoja apuntado por A1 (c) que contendrá, además del objeto 3a, los objetos 1 y 2, de esta forma aun cuando sólo un objeto cambia su posición, el camino entero puede necesitar ser duplicado.

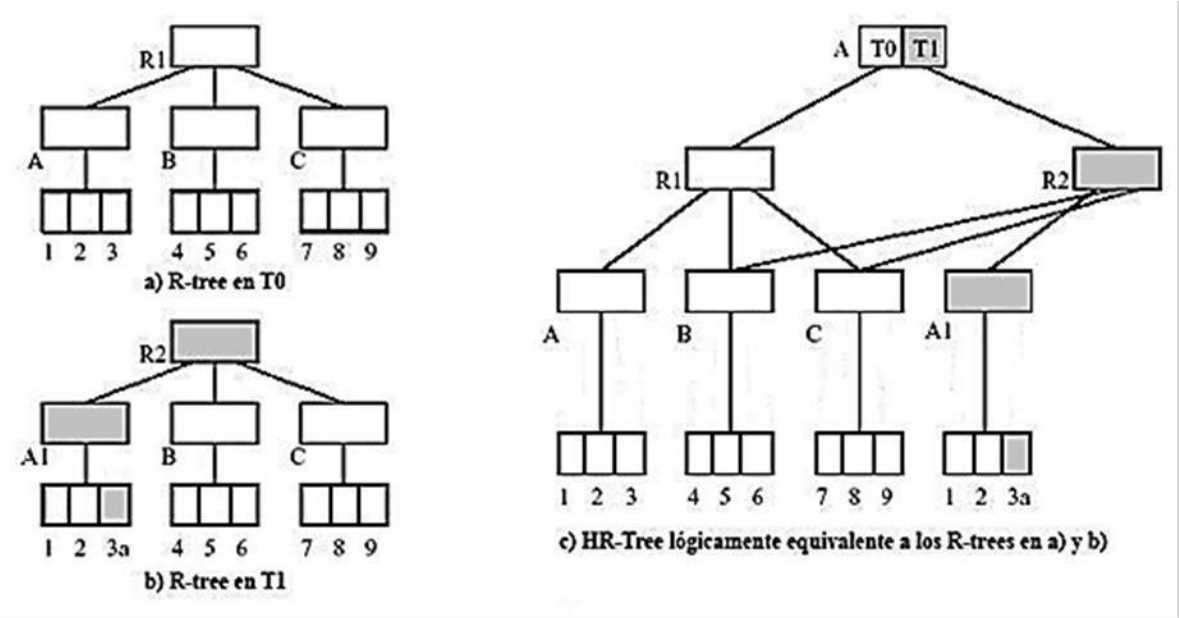


Ilustración 5: Hr-Tree.

2.1.5. Mvr-Tree

El Mvr-Tree (Tao y Papadias, 2000) es un tipo de estructura que maneja multi versiones en que cada entrada es de la forma *<S, t_{start}, t_{end}, puntero>*, donde *S* representa el valor del atributo espacial de un objeto para los nodos hojas o el MBR (Mínimum Bounding Rectangle) para los nodos intermedios (Este MBR incluye a todos los objetos vigentes del subárbol al cual el nodo intermedio hace referencia), t_{start} corresponde al tiempo en que el registro fue insertado en la base de datos, t_{end} el tiempo en que fue borrado y *puntero* apunta al nodo del nivel siguiente en la estructura (nodos intermedios).

Otra característica heredada de MVB-Tree es el uso de múltiples R-Trees para organizar la información, además del concepto de condición de versión débil, que garantiza que el número de entradas vivas (registros que no han sido eliminados de la base de datos) en cualquier instante de tiempo es *0* o al menos $b \cdot P_{version}$, donde *b* es la capacidad del nodo y $P_{version}$ es un parámetro del árbol.

2.1.6. Mv3r-Tree

El Mv3r-Tree (Tao y Papadias, 2000) fue pensado para solucionar las deficiencias que presentan algunos métodos sobre consultas de tipo Time-Slice o Time-Interval, combinando dos estructuras: un MVR-Tree y un 3D R-Tree para los nodos hojas.

Del análisis a los métodos de acceso se puede concluir que Mv3r-Tree resulta el mejor evaluado tanto para consultas de tipo Time-Slice como Time-Interval.

2.2. Resumen Capítulo

En este capítulo se dio a conocer los arboles más importantes que se usan en las bases de datos espacio-temporales, siguiendo un orden cronológico, teniendo más importancia para nuestro estudio el Mvr-Tree y el Hr-Tree, ya que, como se describió anteriormente, son los índices más idóneos para nuestras consultas.

En el siguiente capítulo se presentarán los conceptos básicos del procesamiento paralelo, cuales usaremos nosotros y su justificación.

Capítulo III

3. Procesamiento Paralelo y Diseño del Algoritmo

El procesamiento paralelo se define como el uso de computadores paralelos con la finalidad de reducir el tiempo requerido para resolver un único problema computacional. Por su parte, un computador paralelo es un sistema de cómputo multi-procesador que soporta programación paralela. Esta última permite explicitar cuantas porciones diferentes de datos serán procesadas paralelamente por los diferentes procesadores.

En la actualidad el procesamiento paralelo permite resolver problemas donde se necesita alta capacidad de cómputo o manejar datos a tiempo real que puedan ocupar grandes cantidades de almacenamiento. Aplicaciones en ingeniería y diseño, aplicaciones científicas y aplicaciones comerciales son algunos ejemplos en donde es posible utilizar paralelismo.

Existen varias arquitecturas que sustentan el procesamiento paralelo, estas fueron propuestas por Michael J. Flynn y son conocidas como la "La Clasificación o Taxonomía de Flynn" (Flynn, 1972). Dentro de estas encontramos:

- **SIMD: Simple Instruction Multiple Data.** Este modelo posee un flujo de instrucciones único y un flujo de datos múltiple, en donde una única instrucción es aplicada sobre diferentes datos al mismo tiempo. Cada procesador dispone de una memoria asociada, de manera que cada instrucción es ejecutada simultáneamente por cada uno de estos, pero sobre un conjunto de datos distintos.

- **MISD: Multiple Instruction Simple Data.** Este modelo posee un flujo de instrucciones múltiples y un flujo de datos único, en donde varias instrucciones trabajan simultáneamente sobre una única partición de datos. Este tipo de sistemas se pueden considerar como máquinas formadas por varias unidades de procesamiento, las cuales reciben diferentes instrucciones que operan sobran los mismos datos o como una clase de máquinas donde un mismo flujo de datos viaja a través de numerosas unidades de procesamiento.

- **MIMD: Multiple Instruction Multiple Data.** Este modelo posee flujo de instrucciones y de datos múltiples, en donde un grupo de unidades de procesamiento ejecuta simultáneamente diferentes secuencias de instrucciones sobre conjuntos de datos diferentes.

Existe una versión restringida de este modelo, denominado SPMD, en donde todos los procesadores ejecutan el mismo programa. Esta Arquitectura se puede subdividir según la forma en que los procesadores se comunican, teniendo así dos clasificaciones:

- **Paralelismo con Memoria Compartida (SMP).** Estos sistemas comparten una memoria en común de esta manera cada procesador accede a los programas y datos almacenados en la memoria global del sistema. Estos sistemas comparten además el sistema de bus y el sistema de entrada/salida, y son controlados por un mismo sistema operativo. En general la programación se hace a través de Threads o Hilos.
- **Paralelismo con Memoria Distribuida.** Estos sistemas poseen una memoria dedicada, así cada elemento de procesamiento es en sí un computador. La interacción entre cada procesador se realiza a través de una red de comunicación. En un sistema cluster típico generalmente está formado por una máquina que actúa como servidor maestro y uno o más máquinas esclavas. Esta arquitectura utiliza para la comunicación entre máquinas, mecanismos de paso de mensajes a través de librerías tales como, PVM[2] o MPI[3].

Para generar un sistema paralelo eficiente es necesario contar las herramientas adecuadas que le den al programador una visión simplificada y transparente de la arquitectura. Es para esto que se han ideado una serie de modelos de programación que permiten la interacción entre los procesadores participantes en un sistema paralelo. Dentro de los principales modelos de programación encontramos (Foster, 1995):

- **Modelo de Paralelismo sobre Datos.** El término paralelismo sobre datos se refiere a la concurrencia obtenida cuando la misma operación es aplicada a algunos o a todos los elementos de un conjunto de datos. Un programa paralelo sobre datos es una secuencia de tales operaciones. Un algoritmo paralelo se obtiene de un programa paralelo sobre datos mediante la aplicación de técnicas de descomposición del dominio a las estructuras de datos sobre las que se operan.

[2] Herramientas para una red en línea de computadores.

[3] Lenguaje que aporta sincronización entre procesos y permite que exploten la existencia de múltiples procesadores.

Entonces las operaciones se particionan, a menudo de acuerdo a la regla de "el propietario computa", es decir, el procesador que "se adueña" de un valor es responsable de su actualización.

- **Modelo de Memoria Compartida.** En el modelo de memoria compartida, los procesos comparten un espacio de direcciones común en donde leen y escriben asincrónicamente. Existen varios mecanismos, tales como cerrojos y semáforos, que pueden ser usados para controlar el acceso a la memoria compartida (control de concurrencia). Para lograr implementar un programa paralelo en un modelo de memoria compartida, es necesario sincronizar el acceso a los datos para evitar que cada proceso interfiera con el resto de los procesos en la lectura o escritura en memoria. De esta forma se asegura que no se produzcan resultados incorrectos en el acceso a memoria. Esta sincronización puede realizarse mediante, cerrojos, semáforos o barreras.

- **Modelo de Paso de Mensajes.** Los programas que utilizan el paso de mensajes crean múltiples procesos, en donde cada uno se ejecuta en un nodo, con un sólo procesador y una sola área de memoria. Cada proceso se identifica mediante un identificador único, y estos interactúan enviándose y recibiendo mensajes que contienen la información que se requiere intercambiar. Además del intercambio de mensajes entre procesadores, se requiere también sincronización entre los mismos, en el caso de que algún proceso requiera que otros finalicen sus tareas antes de poder continuar. Estos sistemas son adecuados para implementar la arquitectura SPMD (Single Program, Multiple Data) debido a que cada proceso ejecuta el mismo programa, pero trabaja sobre diferentes datos. Existen dos interfaces de Paso de Mensajes, las cuales son:

    - **MPI (Message Passing Interface).** La Interfaz de Paso de Mensajes (conocido ampliamente como MPI) es un protocolo de comunicación entre computadoras. Es el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida.

Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que pueden ser utilizadas en programas escritos en los lenguajes de programación C, C++, Fortran y Ada.

- **BSP (Bulk-Synchronous Parallelism).** El modelo de paralelismo BSP, resulta una propuesta promisoria para establecer los enlaces necesarios entre el hardware y el software paralelos (Pérez, 2003). Los algoritmos programados basados en el modelo BSP, son portables a cualquier arquitectura paralela. El modelo BSP propone además un mecanismo para determinar la complejidad de los algoritmos paralelos. Este mecanismo, al tener en cuenta detalles como la comunicación y la sincronización, permite hallar una complejidad que caracteriza de manera más adecuada el desempeño real del algoritmo.

- Del análisis y estudio de los conceptos de procesamiento paralelo se consideró la escalabilidad como uno de los factores preponderantes en la elección de la arquitectura a utilizar para el procesamiento paralelo de la estructura de datos. Ya que dependiendo de los requerimientos de los experimentos (y de investigaciones futuras) puede ser necesario ejecutar la aplicación en distinto número de procesadores cada vez. Además, la división del dominio del problema hace que existan, en porciones similares de datos para cada proceso, objetos distintos, con lo cual los datos accesados serán diferentes y factibles de manejar en memoria local e individual a cada árbol. Por otro lado, el hecho de que los accesos a memoria son muy frecuentes en el tipo de aplicación con la cual se está trabajando, se requiere el uso de medios de control de concurrencia lo que dificulta la programación en un modelo de memoria compartida.

Debido a estos motivos se ha elegido implementar la aplicación utilizando un cluster de computadores, en donde la aplicación será ejecutada por cada máquina (SPMD), y dividiendo los datos que generarán el árbol en partes similares y distribuidas en la memoria local de cada máquina. Esta implementación se realizará mediante la interfaz de paso de mensajes (MPI), compilando y ejecutando con mpicc y mpirun respectivamente.

3.1. Diseño del Algoritmo Paralelo En Línea.

3.1.1. Estrategia de División del Dominio

Se presentarán cuatro estrategias de división del dominio que fueron propuestas por los docentes a cargo del proyecto "Paralelización de Estructuras de Datos y Algoritmos para Bases de Datos Espacio-Temporales", encabezado por el profesor Claudio Gutiérrez Soto, quien patrocina este trabajo.

Antes de definir cada estrategia es importante destacar algunos elementos que inciden en el desarrollo de cada una. Con el fin de implementar eficientemente cada algoritmo paralelo propuesto, se han considerado los siguientes factores:

- **Distribución Inicial de la Carga**: Las tareas identificadas como paralelizables, deben ser distribuidas sobre los procesadores de forma que se procesen en el menor tiempo posible, asignando tareas de menor complejidad a nodos del cluster con baja capacidad de procesamiento y dejando las tareas de mayor dificultad a los procesadores con mejor desempeño. Dado que la implementación paralela sugerida se encuentra bajo una arquitectura de tipo cluster en donde los nodos que ejecutan las tareas (creación de los árboles y consultas sobre ellos) tienen iguales capacidades de procesamiento, como se mencionó al final del capítulo anterior, se realiza una distribución de carga equitativa de las tareas asignadas a cada procesador. De la misma forma, los datos que manipulará cada nodo en el cluster serán divididos equitativamente, evitando desviaciones en el análisis posterior de las respuestas a las consultas, debido a que el nivel de procesamiento es el mismo para cada nodo y los árboles generados son similares (no se puede asegurar que sean idénticos en forma ya que depende de los datos e incluso del orden en que se ingresen).

- **Balance de Carga**: El balance de carga es una forma de distribuir las tareas entre los procesadores, sin embargo, ésta se realiza de forma dinámica donde los cambios en la distribución se hacen a tiempos de ejecución, es decir, hay migración de tareas en tiempo de ejecución.

- **Comunicación entre Procesadores**: Es necesario evitar la excesiva comunicación entre procesadores con el fin de crear cierta independencia entre ellos, que asegure un buen desempeño en el procesamiento de datos y las propias tareas. Un ejemplo de lo anterior se observa en la dependencia que puede generar la necesidad de un procesador A de acceder a un resultado obtenido en otro proceso B. A quedará a la espera del resultado de B mientras este último no termine de procesar, perdiendo tiempo de procesamiento que podría estar destinado a cumplir otras tareas.

3.1.2. Estrategia Local de Paralelización

Para N datos y P procesadores, el objetivo de esta estrategia es crear árboles independientes en cada máquina, asociando N/P datos a cada una, como se muestra en la ilustración 6.

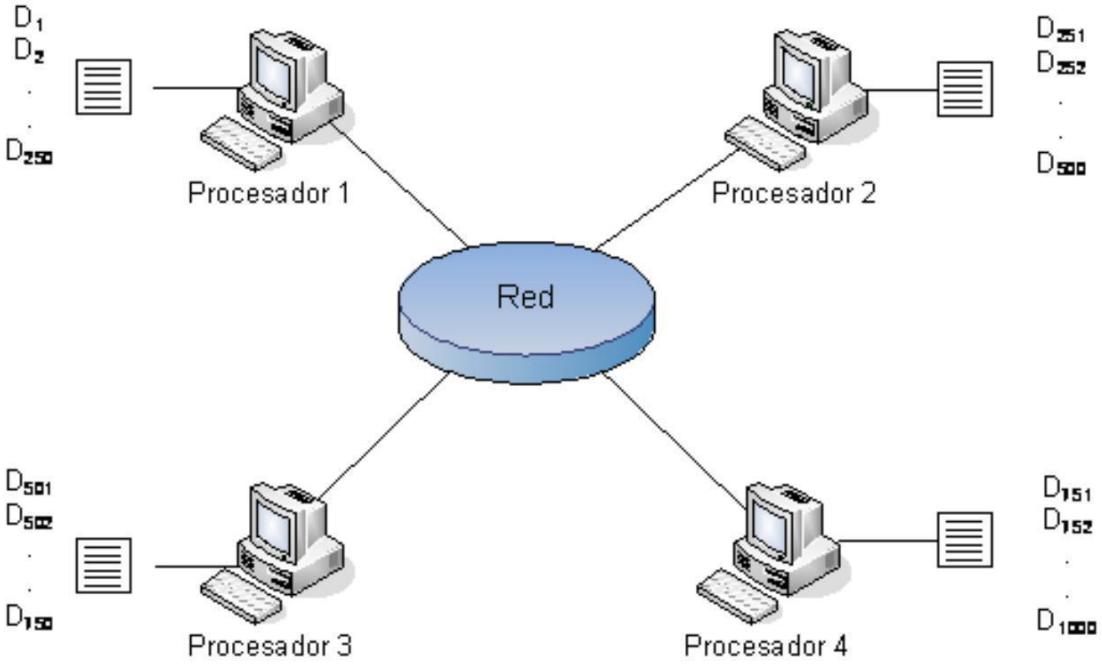


Ilustración 6: Ejemplo de distribución de datos para N=1000 y P=4.

3.1.3. Estrategia de Distribución Circular

Este método se asemeja a la estrategia local en la distribución de datos, la diferencia es que no se divide el archivo de datos original en P partes, sino que se toma cada dato del archivo y se asigna a un procesador. De esta forma, dado P procesadores, el dato 1 será asignado al procesador 1, el dato 2 al procesador 2, el dato i al procesador P, el dato i+1 al procesador 1, el dato i+2 al procesador 2, y así sucesivamente hasta repartir todo el conjunto de datos, como se muestra en la **Ilustración 7**.

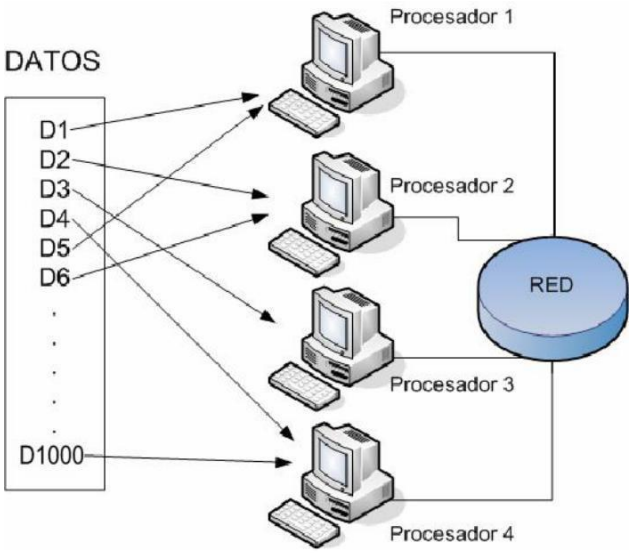


Ilustración 7: Asignación de datos en Estrategia de Distribución Circular para P=4 y N=1000.

3.1.4. Estrategia de un Algoritmo En Línea

Nuestra estrategia se basa en balanceo de carga en los procesadores por medio de un algoritmo en línea, la base principal de este algoritmo es que los objetos modelados en la base de dato Espacio-Temporal no tienen la misma movilidad, por lo tanto, si un objeto se movió, probablemente éste lo hará nuevamente. Una distribución homogénea de los objetos que más cambien su posición en los procesadores debería dar como resultado el ahorro el balanceo de cargas disminuyendo el tiempo de procesamiento de las consultas.

3.1.5. Descripción del Algoritmo En Línea

Lo que hace el algoritmo es identificar y organizar los objetos según su movilidad en: móviles y estáticos, esto se realiza identificando en primera instancia aquellos objetos que cambiaron su posición al pasar de un time-stamp a otro, estos se guardarán como móviles y los restantes objetos serán guardados como estáticos (líneas 3 y 4, **ilustración 10)**. Una vez organizados los objetos se procede a realizar la distribución circular según la cantidad de procesadores. Para esto se tomará la lista de objetos móviles y se irán repartiendo de a uno en cada procesador (como se muestra en la **ilustración 8**) de manera que en el procesador número 1 irá quedando el primer objeto de la lista móviles, en el procesador número 2 el segundo objeto de la lista y así sucesivamente hasta terminar la lista de objetos móviles, finalmente se realizará el mismo proceso anterior pero esta vez con los objetos estáticos (como se muestra en la **ilustración 9**) logrando un buen balanceo de carga en todos los procesadores.

Al mismo tiempo este algoritmo permite identificar todas aquellas consultas que interactúan en línea con los objetos mencionados anteriormente, permitiendo contar la cantidad de consultas Time-Slice y Time-Interval, conocer las diez consultas que más se ejecutan, y, conociendo estos datos, permitirá identificar el índice más óptimo, siendo Mvr-Tree para consultas Time-Interval y Hr-Tree para consultas Time-Slice (línea 5 hasta 27, **ilustración 10**).

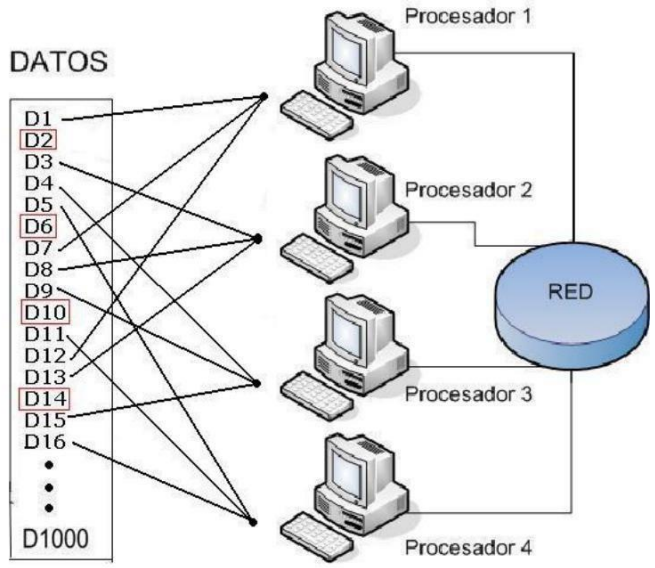


Ilustración 8: Distribución objetos móviles.

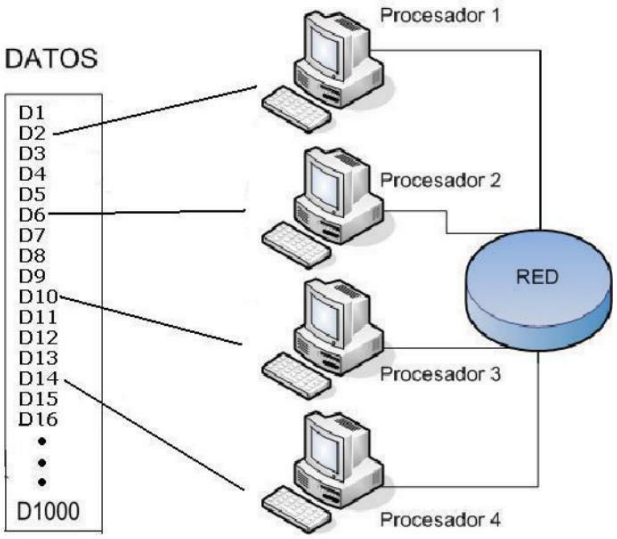


Ilustración 9: Distribución objetos estáticos.

```
Procedure ALGORITMO
        Si t = t', t' un instante particular del tiempo.
                Reviso los objetos móviles.
          Reviso los objetos estáticos.
          Reviso los tipos de consultas más comunes.
IF (Tipo de consulta = Time-Interval && índice actual = R-Tree)
{
          Cambio índice a Mvr-Tree
          Distribuyo los objetos móviles con distribución circular
          Distribuyo los objetos estáticos con distribución circular
}
else IF (Tipo de consulta = Time-Interval && índice actual = Mvr-Tree)
{
          Distribuyo los objetos móviles con distribución circular
          Distribuyo los objetos estáticos con distribución circular
}
IF (Tipo de consulta = Time-Slice && índice actual = R-Tree)
{
          Cambio índice a Hr-Tree
          Distribuyo los objetos móviles con distribución circular
          Distribuyo los objetos estáticos con distribución circular
}
else IF (Tipo de consulta = Time-Slice && índice actual = Hr-Tree)
{
          Distribuyo los objetos móviles con distribución circular
          Distribuyo los objetos estáticos con distribución circular
}
```

Ilustración 10: Seudo Código Algoritmo

3.1.6. Objetos

Para el diseño y construcción del algoritmo, se necesita detallar todos los archivos que se relacionan, los cuales se especificarán a continuación.

En el algoritmo hay dos conceptos claves, por una parte, están los objetos y por otro las consultas.

Los objetos se guardan en un archivo llamado “SET”, los cuales, se moverán con una distribución de probabilidad Zipf, permitiéndonos movimientos al azar en distintos Time-Stamp. Los objetos están representados de la siguiente manera:

Oid Ti X1 Y1 X2 Y2.

Donde:

- **Oid:** Identificador del Objeto.
- **Ti:** Tiempo de Inserción (Time Stamp).
- **X1:** Coordenada X de la esquina inferior izquierda del MBR.
- **Y1:** Coordenada Y de la esquina inferior izquierda del MBR.
- **X2:** Coordenada X de la esquina superior derecha del MBR.
- **Y2:** Coordenada Y de la esquina superior derecha del MBR.

En la **ilustración 11** se puede ver una representación gráfica de los objetos.

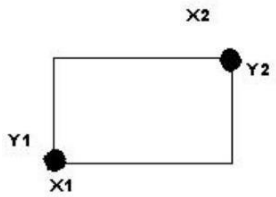


Ilustración 11: Ilustración MBR.

Para la simplificación de los estudios, se igualaron las coordenadas, es decir X1=X2 y Y1=Y2.

En este archivo set, se harán todas las operaciones necesarias que se describieron en los capítulos anteriores, las cuales son, obtener móviles, estáticos y crear el árbol de objetos, lo cual nos permitirá ejecutar las consultas en éste. Todo esto se especificará en los puntos de este capítulo.

3.1.7. Consultas

Luego de la creación del árbol de objetos se necesita pasar a la siguiente etapa, la cual es ejecutar las consultas y obtener los nodos recuperados. Pero para eso se necesita especificar dónde y cómo se guardan las consultas.

Las consultas se guardan en un archivo llamado “Consultas.txt” (como se ve en la **ilustración 12**), el cual tiene la siguiente estructura:

X1 X2 Y1 Y2 TI TF

Generando una línea en el mapa, la cual obtendrá todos los objetos que se encuentren sobre ella, donde:

- **X1, Y1:** Punto inicial de la línea.
- **X2, Y2:** Punto final de la línea.
- **TI:** Tiempo inicial del time-stamp.
- **TF:** Tiempo final del time-stamp.

```
0.840187 0.394383 0.783098 0.798439 0 0
0.911646 0.197551 0.335222 0.768229 0 0
0.277774 0.553969 0.477397 0.628870 0 0
0.364784 0.513400 0.952229 0.916194 0 0
0.635711 0.717296 0.141602 0.606968 0 0
0.016301 0.242887 0.137231 0.804176 0 1
0.156679 0.400944 0.129790 0.108809 0 1
0.998923 0.218257 0.512932 0.839111 0 1
0.612639 0.296031 0.637552 0.524287 0 1
0.493582 0.972774 0.292516 0.771357 0 1
```

Ilustración 12: Archivo Consultas.txt

3.1.8. Creación Archivo Set

Para la creación del archivo "set" se necesita comprender todo lo explicado en el punto anterior, es decir, para la creación se necesita crear una lista ligada por punteros (como muestra la **ilustración 13**), donde cada lista será un objeto del set. Este código se implementó en el archivo "Proyecto.C" y se puede entender a través del seudo código mostrado en la **ilustración 14**.

```
struct objeto{

int oid;

float  ti;

float x1;

float y1;

float x2;

float y2;

int p;

struct objeto *next;};
```

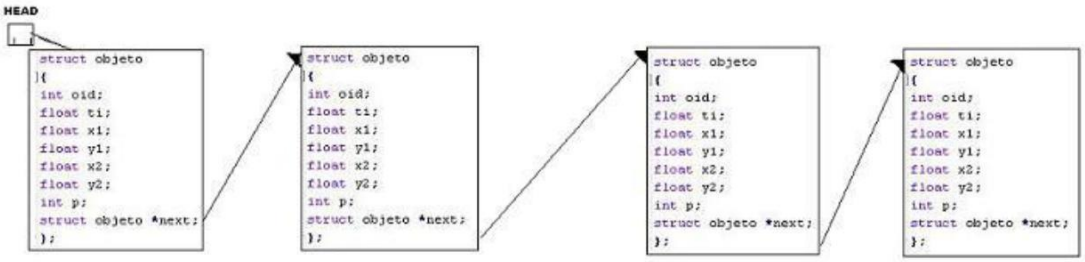


Ilustración 13: Lista Ligada por Punteros

Procedure Crear Archivo Set
Ingreso Cantidad de Objetos del Set.
Ingreso Cantidad de Time-Stamp del Set.
Ingreso Valor Alpha para distribución Zipf.
Ingreso Valor N de distribución Zipf, el cual nos permite dar un rango a los números aleatorios.
Ingreso de valor num value de distribución Zipf, el cual nos permite dar la cantidad de números aleatorios.
Ingreso de Cantidad de objetos que se moverán en el Set (número par).
Ciclo For para crear el primer Time-Stamp en el archivo Set.
Ciclo For para crear los siguientes Time-Stamp en el archivo Set.
En este ciclo For, los objetos de moverán con distribución Zipf, es decir, que a cada coordenada se le suma un valor dependiendo del número que arroje la distribución.

Ilustración 14: Seudo Código para Crear Set.

3.1.9. Leer Objetos

Función para mostrar por pantalla los objetos creados, imprimiendo toda la estructura del objeto.

Procedure Leer Archivo Set
Lee el Archivo Set generado en la función Crear Set.
Lee línea por línea nuestro Archivo Set.
Guarda en una lista las variables.
Recorre la lista y la imprime por pantalla.

Ilustración 15: Seudo Código para Leer Objetos.

3.1.10. Obtención de objetos móviles, estáticos y estadísticas de los objetos

Como se puede ver en el seudo código de la **ilustración 16** para encontrar cuál(es) de los objetos del dataset se movieron y/o cambiaron de forma, es que utilizaremos una lista ligada por punteros de estos datos. Creamos un archivo para guardar los objetos más móviles, si encontramos el mismo oid y que las coordenadas son distintas (o la X o la Y) quiere decir que nuestro objeto se ha movido y/o cambiado de forma, por lo tanto, lo guardamos en el archivo "Móviles".

Una vez que tenemos el archivo "Móviles", podemos buscar a los objetos "estáticos" (aquellos objetos que no cambiaron de posición), serán aquellos objetos que se encuentren en el dataset, pero que no se encuentren en el archivo "móviles".

La estructura de las listas y el ejemplo de un archivo generado se pueden ver en la **ilustración 17.**

Procedure Generar Estadísticas A Partir Del Archivo Set Lee
Archivo Set generado en la función Crear Set. Crea
Lista del Archivo Set.
Crea Archivo "Móviles".
Compara lista original con auxiliar, comparando las coordenadas en los distintos Time.Stamp, si son distintas (Se movieron) se guardan en el archivo Móviles.
Lee Archivo Móviles generado.
Crea Lista a partir del Archivo móviles.
Crea Archivo "Estáticos".
Compara lista de archivo móviles con lista de archivo Set, si no se encuentra quiere decir que el objeto es estático. Guardando este objeto en el archivo "Estáticos".
Se genera lista con archivo "Estáticos".
Se imprime por pantalla la cantidad de objetos estáticos y móviles.

Ilustración 16: Seudo Código para Generar Estadísticas.

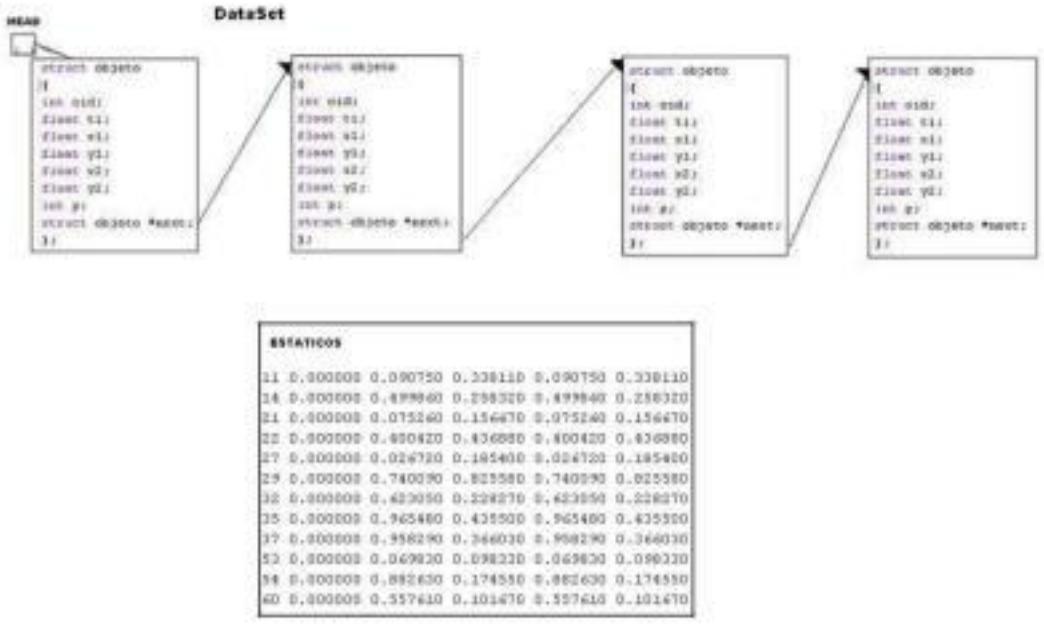


Ilustración 17: Lista Ligada por Punteros de los Archivo Móviles, Data set y archivo Estáticos.

3.1.11. Distribución de Objetos

Como se puede ver en el seudo código de la **ilustración 19** lo que haremos para distribuir primero los objetos que tiene una probabilidad mayor a moverse y/o cambiar de forma, creamos la lista ligada por punteros (como se muestra en la **ilustración 18**) del archivo Móviles el cual distribuimos según sea la necesidad del número de procesadores, este es un ejemplo para 4 procesadores, abrimos 4 archivos nuevos, y en él distribuimos de forma circular los objetos con una mayor probabilidad de moverse y/o cambiar de forma en el tiempo del dataset. Luego hacemos lo mismo con el Archivo de los objetos Estáticos.

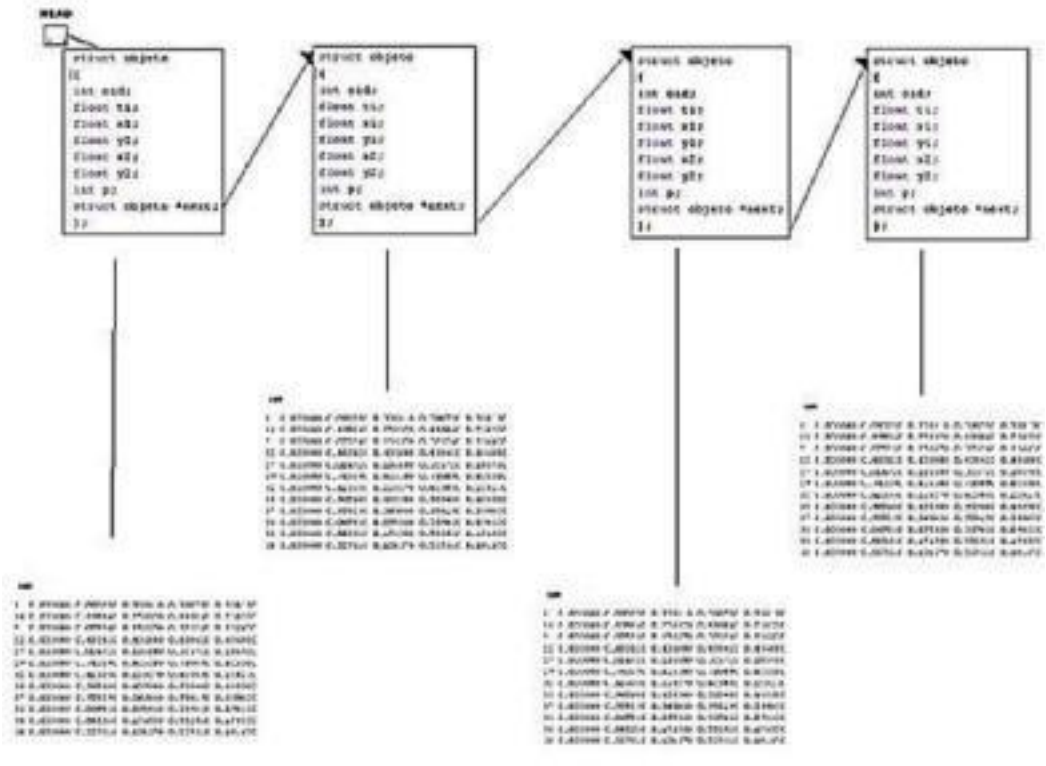


Ilustración 18: Distribución Lista Ligada por Punteros del Archivo Móviles, 4 Archivos/4 Procesadores.

Procedure DISTRIBUCIÓN DE OBJETOS

Lee el Archivo Móviles generado en la función Generar Estadísticas.
Entra a un Switch, con casos para distribuir en 2, 4, 8 y 16 procesadores.
Crea una Lista con el archivo Móviles.
Crea N archivos (dependiendo de la opción que se eligió en el switch).
Guarda en estos archivos los objetos móviles de la lista, guardando el primer objeto en el primer archivo, el segundo objeto en el segundo archivo, y así sucesivamente.
Lee el archivo Estáticos, crea la lista y los guarda con la misma lógica que los objetos móviles.

Ilustración 19: Seudo Código para Distribuir los Objetos.

3.1.12. Creación de Consultas

Para la creación de una consulta se debe crear la siguiente estructura:

```
struct consulta
{
float x1;
float y1;
float x2;
float y2;
int ti; int
tf; int
type;
struct objeto
*next;};
```

Luego de crear la estructura se creará una lista ligada por punteros, donde cada lista será una consulta en el archivo "Consultas.txt".

Procedure GENERADOR DE CONSULTAS
Ingreso Cantidad de Consultas.
Mediante un Ciclo For, se generan consultas con coordenadas entre 0 y 1 al azar.
Se guardan las coordenadas, tiempo inicial y tiempo final en el Archivo Consultas.txt.

Ilustración 20: Seudo Código para Generar Consultas.

3.1.13. Código Mvr-Paralelo

El código ocupado para análisis fue el diseñado por (Tao y Papadias, 2000), y se creó bajo una arquitectura Windows en C++. Este código es compilado y ejecutado en ambiente Linux, y consta de un archivo "mvr_paralelo" en el que se encuentran los llamados a funciones del MVR-Tree. Particularmente existen 2 funciones en tal archivo, las correspondientes a la creación del árbol y la realización de consultas sobre él.

- **Creación del Árbol:** Los datos de los objetos que se quieren insertar y manipular se encuentran en la carpeta “data_set”. Mediante la función “test5( )” del archivo main.cpp, se crea el árbol llamando a la función “BuildMVRTree( )” a la cual se le entrega por parámetros la ruta y nombre del archivo que contiene los datos y los nombres y rutas de los archivos que guardarán las estructuras del árbol con los datos a procesar, (en este caso particular, las carpetas con los árboles creados se llaman “tree” más respectivo número del procesador).

- **Consultas Sobre el Árbol:** La función “test1( )” del archivo mvr_paralelo realiza las consultas sobre el árbol. En la carpeta “query_set” se encuentra las consultas para el árbol creado. Se parte cargando en memoria principal el árbol guardado anteriormente sobre el que se harán las consultas y con la función “MVRTimeQuery( )” se realizan las consultas definidas en el archivo que debe entregarse como parámetro. El resultado de las consultas se entrega por pantalla y define el número de consultas ejecutadas, el número de registros recuperados para esa consulta y el número de bloques accesados (representados por operaciones de entrada/salida) que fueron necesarios para recuperar tales registros.

3.1.14. Detalles Generales Implementación Paralela

Fueron guardados en la carpeta “data_set” con nombres “set1”, “set2”, “set3”, “set4”, “setN” siendo N referencia a la máquina que ocupará cada conjunto de datos. De la misma forma se identifican las carpetas que contendrán los árboles con los objetos de cada procesador. Es necesario aclarar que el acceso al cluster se realiza a través de una cuenta en el nodo maestro, host “cluster.cienciasbasicas.cl”, y que los datos que se manipulan en este nodo son replicados en el nodo de cómputo del clúster.

Debido a esto el manejo de los datos se realiza solo en el nodo maestro y el acceso a ellos se determina por código a través de los identificadores de los nodos que entregan las funciones de la librería MPI, específicamente MPI_Comm_Size( ) y MPI_Comm_Rank.

Como el código original del MVR-Tree fue creado usando C++, la compilación con MPI debe realizarse a través del compilador de C++ de MPI, mpiCC, logrando una compilación exitosa en la combinación de C para el archivo principal y C++ para las librerías de creación de los árboles y la ejecución de consultas. La ejecución del programa se realiza con el comando mpirun, fijando el número de procesadores ocupados a 4 ,8 y 16 en esta implementación.

3.1.15. Detalles Codificación

Para la implementación de la estrategia local de paralelización se utilizan dos archivos de código fuente, test.cpp y mvr_paralelo.c, en donde se concentra y codifica toda la algoritmia que hace posible la Paralelización.

- **Archivo "Test.cpp":** En este archivo se realizan aquellas modificaciones sobre el archivo main.cpp del MVR-Tree original, específicamente en las funciones test5() y test1() del código, que permiten paralelizar la aplicación. En el caso de la función test5(), que permite la creación del árbol, la primera modificación que se realiza es la recepción del parámetro "maq" que es un valor que permite identificar la máquina que está ejecutando la aplicación. Este valor es de suma importancia, ya que de acuerdo a la máquina que está operando sobre el código se crea el árbol a partir de los archivos "set1", "set2", "set3" o "set4" según corresponda. De esta forma, cada máquina crea su propio árbol y lo almacena en su carpeta correspondiente, "tree1" para la "maquina 1", "tree2" para la "maquina 2", etc. La creación de árbol se realiza mediante la función "BuildMVRTree". En la función test1() que permite realizar diversas consultas sobre el árbol ya creado en test5(), se realiza también la recepción del parámetro "maq", para después identificar la máquina que está ejecutando el código y cargar el árbol que le corresponda en memoria principal.

  Luego, se realizan las consultas sobre el árbol en cada una de las máquinas mediante la función "MVRTimeQuery" utilizando el conjunto de consultas que se encuentran en carpeta "query_set", para finalmente entregar las respuestas a las consultas realizadas.

- **Archivo "mvr_paralelo.c":** En este archivo se encuentra la función principal del programa, main(), en la cual se inicializan las funciones de paralelización de MPI, mediante "MPI_Init", se recupera la cantidad de máquinas o procesos participantes en el procesamiento paralelo y se le asigna el identificador a cada máquina, a través de las funciones "MPI_Comm_size" y "MPI_Comm_rank" respectivamente. Luego se invocan test5() y test1() del archivo test.cpp, para así comenzar a crear los árboles y realizar las consultas sobre estos de forma paralela.

3.2. Resumen Capítulo

En este capítulo se describió todo lo relacionado con procesamiento paralelo, definiendo el modelo de paso de mensajes (MPI) junto a la red de computadores, para el desarrollo de nuestro algoritmo.

También se define la forma de distribuir las cargas en los procesadores, dando detalle de distintas estrategias para realizarlo, eligiendo la distribución circular de objetos móviles y estáticos, siendo los móviles los con más prioridad.

Otra de las explicaciones fue como crear consultas y cómo funcionan e interactúan con los objetos.

En el siguiente capítulo se profundizará la construcción del diseño experimental, detallando paso a paso todo lo relacionado con el escenario experimental.

Capítulo IV

4. Diseño de Escenario Experimental

4.0. Tipos de comunicación

4.0.1. Comunicación Asincrónica

El primer nodo es el proceso raíz, y tiene la copia inicial de los datos. Todos los demás nodos reciben la copia de los datos

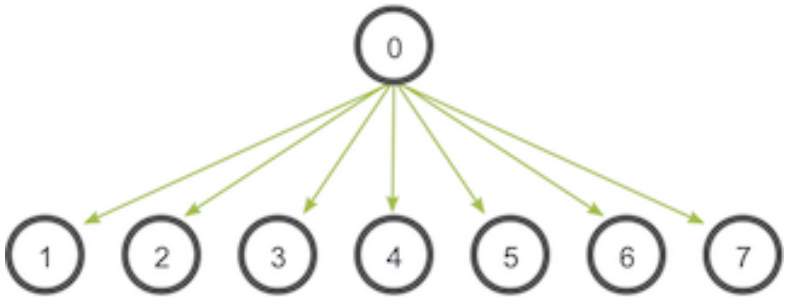


Ilustración: Paso de datos comunicación asincrónica

4.0.2. Comunicación Sincrónica

Los datos van pasando todos a través del nodo, y una vez que todos pasan por el, se continua en el siguiente.

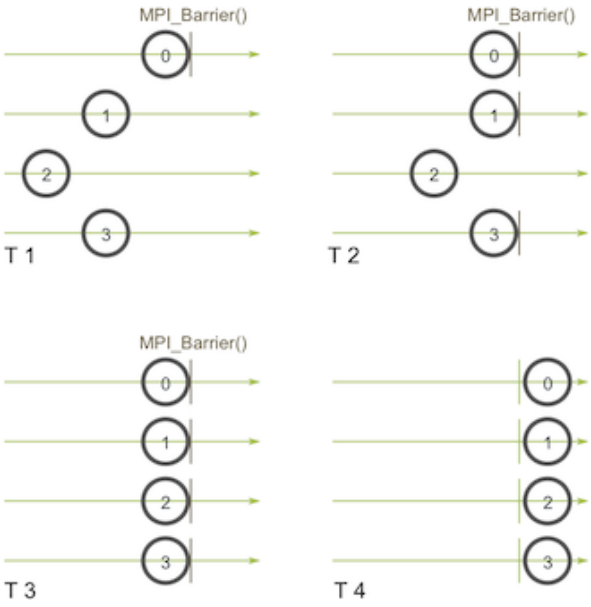


Ilustración: Paso de datos comunicación sincrónica

4.1. Clúster

El término clúster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Hoy en día desempeñan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno. La tecnología de clústers ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software de misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

En palabras simples un clúster es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio. Los clústers son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador, siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

El algoritmo fue ejecutado sobre un clúster, el cual se encuentra en las dependencias de la universidad del Biobío, y el cual cuenta con ocho nodos, donde en el primero de ellos se ejecutó el algoritmo y desde el cual se hace el envío de datos a los demás nodos para así obtener los tiempos asociados a cada experimento.

El clúster utilizado cuenta con la siguiente configuración de hardware asociado:

Servidor	Gabinete microlab
Procesador	Intel ® core™ i7-2600 CPU @ 3.40 GHz
Memoria	16 GB DDR3 1333MHz PC3-10600 non ecc c19 DIMM
Almacenamiento	Wd5000aaks Disco Duro 1 TERA Wester Digital
Sistema Operativo	Linux Debian Squeeze versión 7.8 – 64 Bits
Servicios	Open JDK Server VM (build 14.0-b16, mixed mode) Open JDK Runtime Environment (icedtea 6 1.8.13) Java versión “1.6.0_18” Servidor SSH
Entorno de desarrollo **Programas MPI** **Servicio**	Built Essential Openmpi-bin openmpi-common libopenmpi1.3 libopenmpi-liv Torque-server torque-scheduler torque-client torque-mon
Asignaturas	Programación paralela Programación en Java Asignaturas Magister Proyectos de títulos y tesis

4.2. Distribuciones

Para comenzar con el diseño del escenario experimental es necesario hacer una breve introducción y explicación a las distribuciones de probabilidad a usar, para el escenario se utilizarán tres distribuciones, las que permitirán generar números y darles un uso adecuado.

4.2.1. Distribución Zipf

La distribución zipf permite generar números cuya frecuencia de aparición estará dada por la fórmula de la **ilustración 21**.

$$P_n \sim 1/n^a$$

Ilustración 21: Frecuencia de aparición de los números a generar

Donde Pn representa la frecuencia de una consulta n-ésima y el exponente a es próximo a 1. Esto significa que la segunda consulta se repetirá aproximadamente con una frecuencia de 1/2 respecto a la primera, y la tercera con una frecuencia de 1/3 con respecto a la segunda y así sucesivamente.

4.2.2. Distribución Gamma

Este modelo es una generalización del modelo Exponencial ya que, en ocasiones, se utiliza para modelar variables que describen el tiempo hasta que se produce p veces un determinado suceso.

4.2.3. Distribución Exponencial

La distribución exponencial es una distribución especialmente importante en estadística. Sirve para modelizar el tiempo que transcurre entre dos eventos independientes, y durante los cuales transcurren, por término medio, el mismo tiempo. También se la reconoce por ser la otra cara de la moneda de los procesos de Poisson.

4.2. Diseño de Escenario Experimental

Ya visto todo lo relacionado con el algoritmo en el capítulo tres, comenzamos a explicar cómo se realizaron los experimentos.

Lo primero a explicar está relacionado con los objetos. Como se dijo anteriormente, los objetos se moverán con un 10% de movilidad, teniendo en total 23264 objetos, lo cual aumenta debido a la repetición de estos en los 10 time-stamp generados, lo que nos generó un archivo set (donde se guardan los objetos) de 46521 objetos.

En el caso de las consultas, se crearon 3 archivos de consultas, que se describirán a continuación:

- 50% Time-Slice / 50 % Time-Interval.
- 50 % Time-Interval / 50 % Time-Slice.
- Tipos Intercalados, (1 Time-Slice/ 1 Time-Interval).

Luego de crear las consultas, se generaron números a partir de tres distribuciones probabilísticas, Zipf-Gamma-Exponencial. Números que fueron generados entre 1-100 (Número de consulta), lo cual permitió simular un algoritmo en línea. Esto quiere decir que se generarán 1000 números (siguiendo estas distribuciones) entre 1-100 por cada experimento, lo qué simulará que consultas se ejecutarán. Por ejemplo, al experimento 50% Time-Slice / 50% Time-Interval, se generarán números siguiendo la distribución Zipf, luego con esos números se escogerán las consultas a ejecutar, de la misma forma se realizará con la distribución Gamma y Exponencial.

Después de la creación de las 1000 consultas, el experimento, continua con la ejecución de éstas, dándole un parámetro al algoritmo que permitió saber en qué momento revisar qué tipo de consulta se repite más, este parámetro tomó los valores 300,500 y 700. Es decir, si le entregamos el valor 700, el algoritmo ejecutará las 700 primeras consultas, guardando en un archivo todas las consultas ejecutadas junto a los nodos retornados (tipo caché), además conjuntamente obtendrá l Time-Slice os tiempos comunicación asociados a cada experimento, cuando llegue a la consulta 700 el algoritmo será capaz de saber si se cambia el índice a usar, además de generar un ahorro gracias al archivo caché que se creó.

4.3. Resultados

Tal como se explicó anteriormente, se usaron distintas distribuciones de probabilidad para generar las consultas a ejecutar, en el caso de la distribución exponencial se usarán dos parámetros, los cuales son 0.5 y 1. En el caso de la distribución gamma solo se usó con parámetro 1, y en la distribución zipff se usaron dos, el parámetro 1 y 2.

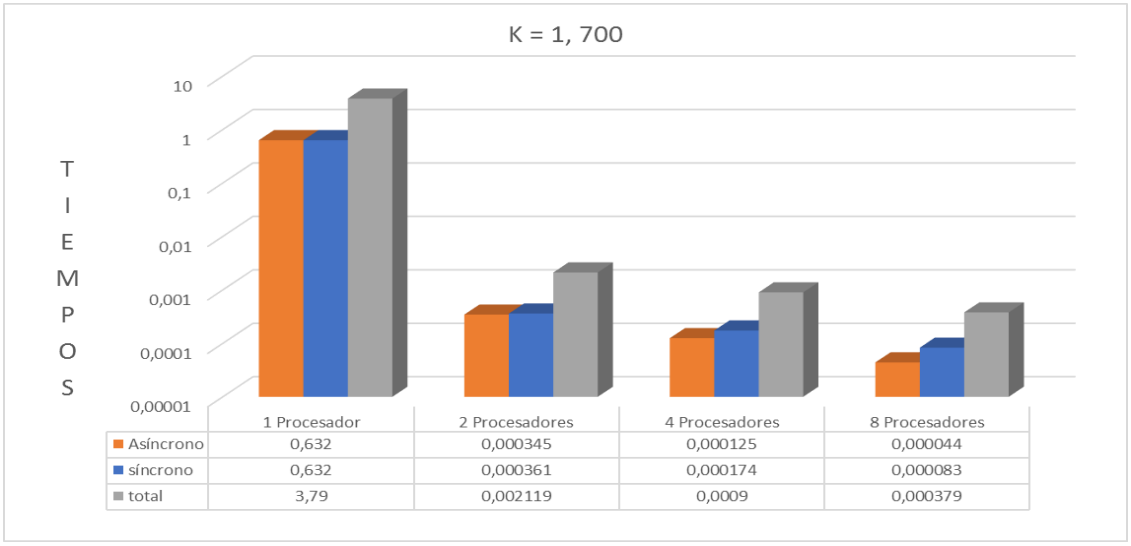
Estos gráficos nos permitirán observar la comparación de tiempos de comunicación por experimento y por cantidad de procesadores.

4.3.1. Distribución Exponencial

- Parámetro: 1.
- Revisar cambio de índice: a la consulta 700.
- Movilidad objetos: 10%.

4.3.1.1. Experimento 1.1 50 TS/50 (700)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

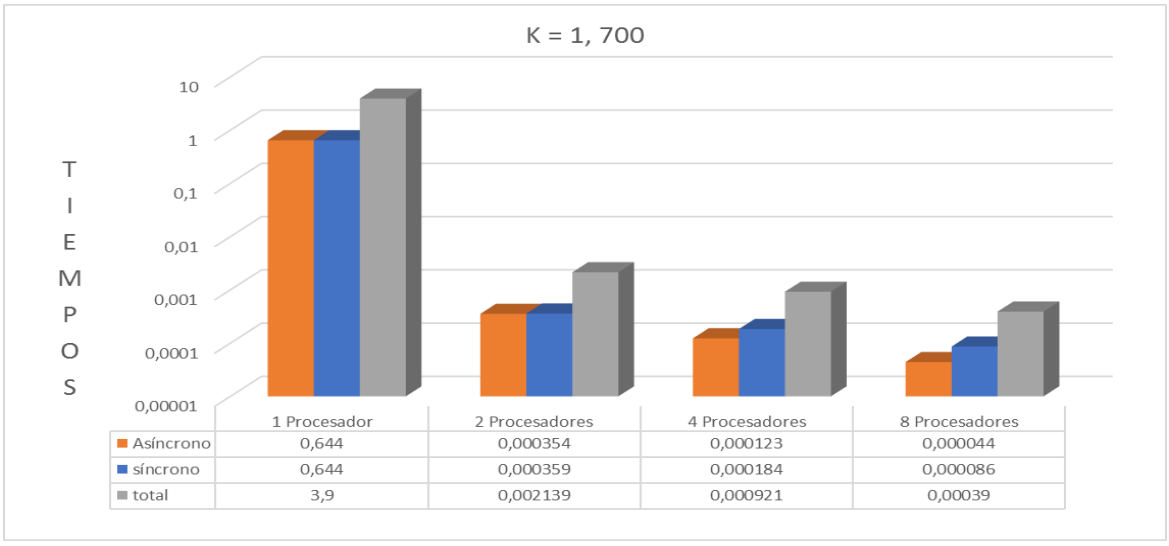


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,632	0,000345	0,000125	0,000044
síncrono	0,632	0,000361	0,000174	0,000083
total	3,79	0,002119	0,0009	0,000379

Ilustración 22: experimento 1.1, exponencial = 1, revisión de consulta = 700

4.3.1.2. Experimento 2.1 50 TI/50 TS (700)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

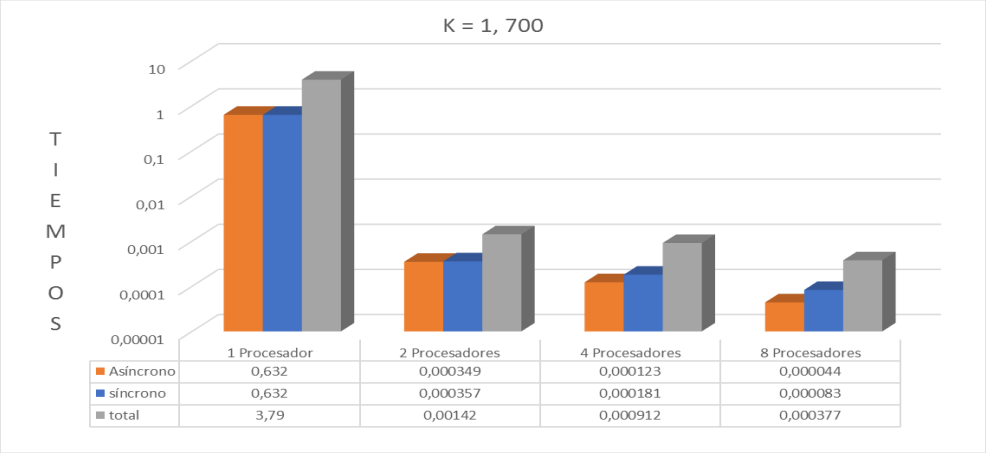


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,644	0,000354	0,000123	0,000044
síncrono	0,644	0,000359	0,000184	0,000086
total	3,9	0,002139	0,000921	0,00039

Ilustración 23: experimento 2.1, exponencial = 1, revisión de consulta = 700

4.3.1.3. Experimento 3.1 1 TI/1 TS (700)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



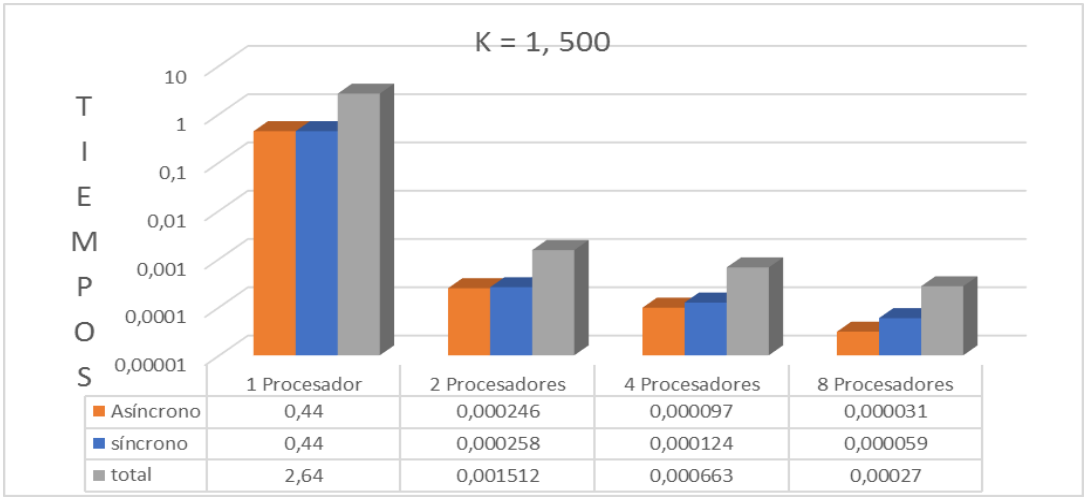
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,632	0,000349	0,000123	0,000044
síncrono	0,632	0,000357	0,000181	0,000083
total	3,79	0,00142	0,000912	0,000377

Ilustración 24: experimento 3.1, exponencial = 1, revisión de consulta = 700

A continuación, se cambiará la revisión del algoritmo a la consulta 500, quedando los tiempos de comunicación de la siguiente manera:

4.3.1.4. Experimento 1.1 50 Ts/50 Ti (500)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

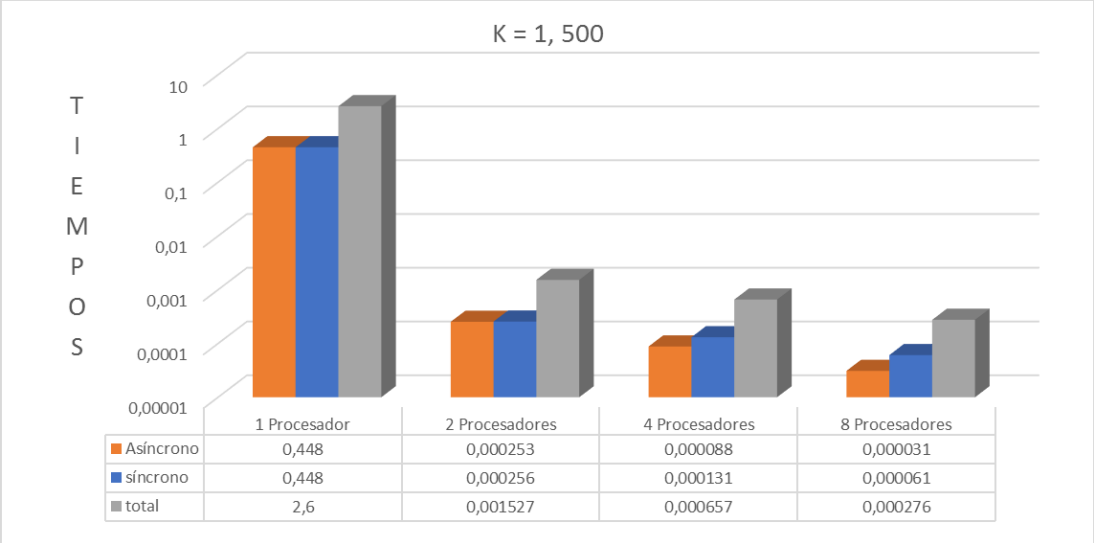


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,44	0,000246	0,000097	0,000031
síncrono	0,44	0,000258	0,000124	0,000059
total	2,64	0,001512	0,000663	0,00027

Ilustración 25: experimento 1.1, exponencial = 1, revisión de consulta = 500

4.3.1.5. Experimento 2.1 50 Ti/50 Ts (500)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

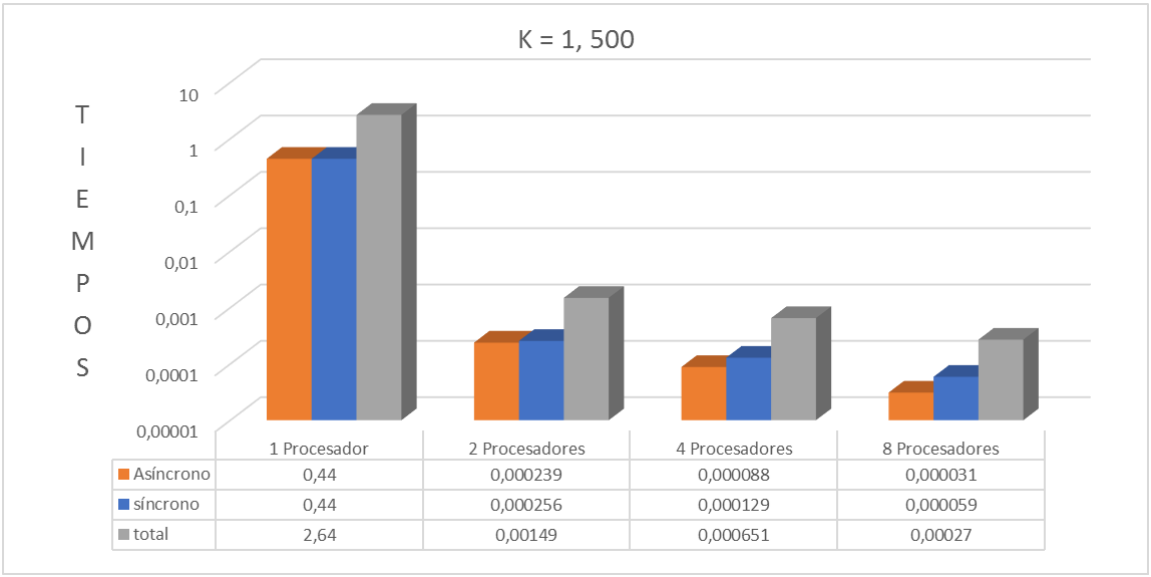


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,448	0,000253	0,000088	0,000031
síncrono	0,448	0,000256	0,000131	0,000061
total	2,6	0,001527	0,000657	0,000276

Ilustración 26: experimento 2.1, exponencial = 1, revisión de consulta = 500

4.3.1.6. Experimento 3.1 1 Ti/1 Ts (500)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



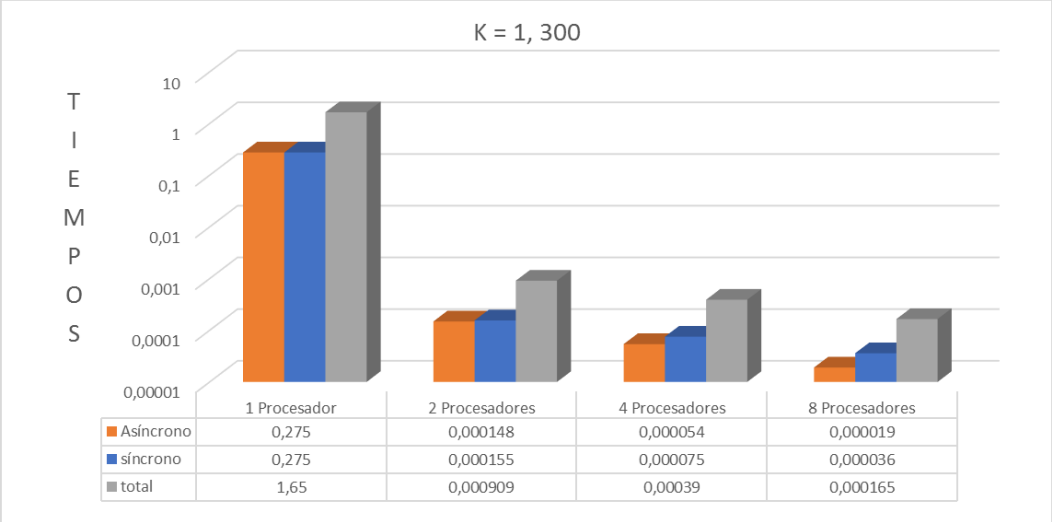
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,44	0,000239	0,000088	0,000031
síncrono	0,44	0,000256	0,000129	0,000059
total	2,64	0,00149	0,000651	0,00027

Ilustración 27: experimento 3.1, exponencial = 1, revisión de consulta = 500

A continuación, se cambiará la revisión del algoritmo a la consulta 300, quedando los tiempos de comunicación de la siguiente manera:

4.3.1.7. Experimento 1.1 50 Ts/50 Ti (300)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

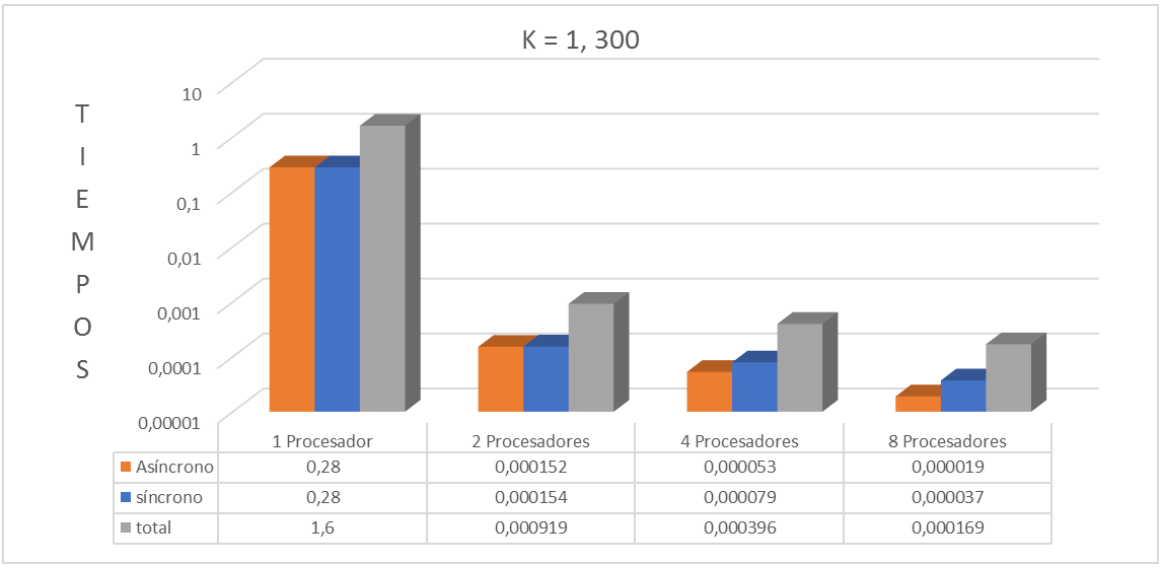


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,275	0,000148	0,000054	0,000019
síncrono	0,275	0,000155	0,000075	0,000036
total	1,65	0,000909	0,00039	0,000165

Ilustración 28: experimento 1.1, exponencial = 1, revisión de consulta = 300

4.3.1.8. Experimento 2.1 50 Ti/50 Ts (300)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

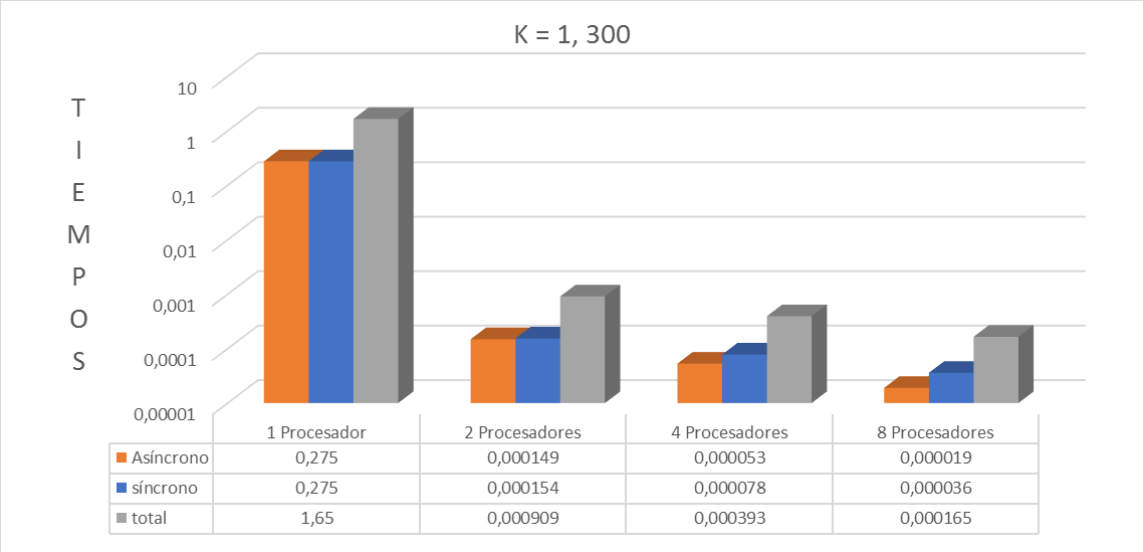


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,28	0,000152	0,000053	0,000019
síncrono	0,28	0,000154	0,000079	0,000037
total	1,6	0,000919	0,000396	0,000169

Ilustración 29: experimento 2.1, exponencial = 1, revisión de consulta = 300

4.3.1.9. Experimento 3.1 1 Ti/1 Ts (300)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



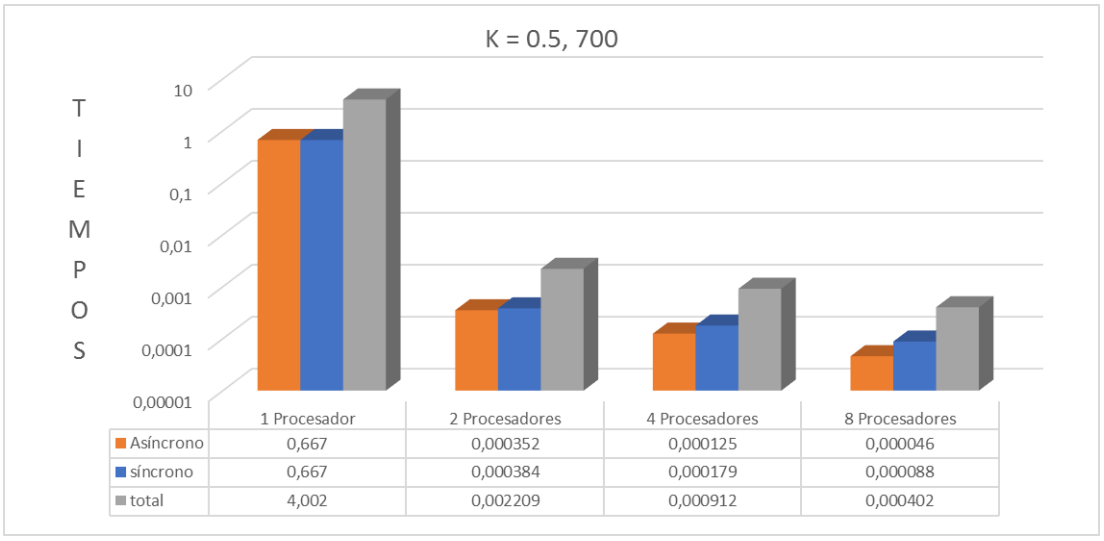
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,275	0,000149	0,000053	0,000019
síncrono	0,275	0,000154	0,000078	0,000036
total	1,65	0,000909	0,000393	0,000165

Ilustración 30: experimento 3.1, exponencial = 1, revisión de consulta = 300

- Parámetro: 0.5.
- Revisar cambio de índice: a la consulta 700.
- Movilidad objetos: 10%.

4.3.1.10. Experimento 1.1 50 TS/50 (700)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

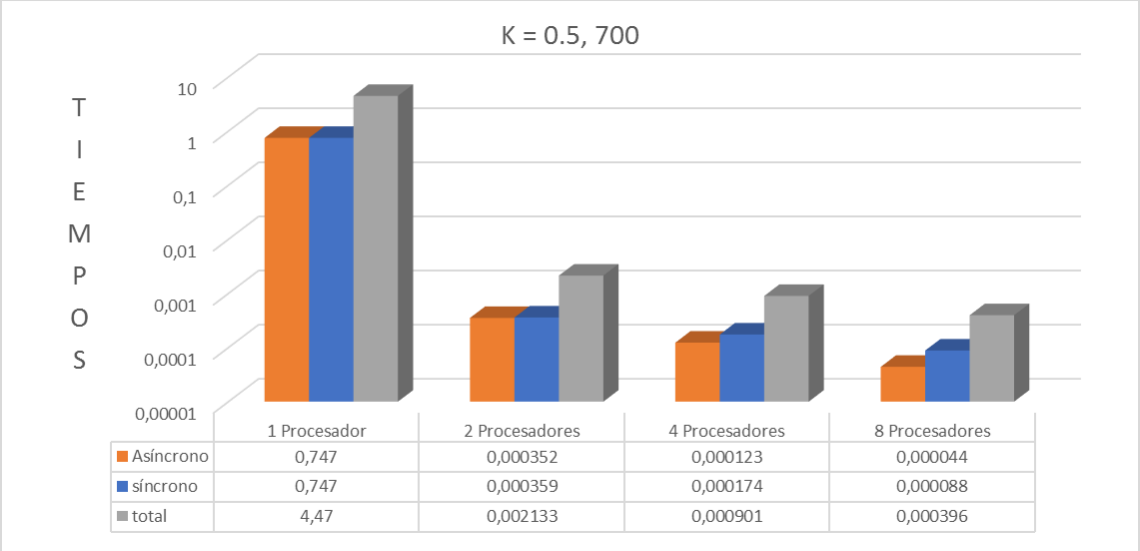


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,667	0,000352	0,000125	0,000046
síncrono	0,667	0,000384	0,000179	0,000088
total	4,002	0,002209	0,000912	0,000402

Ilustración 31: experimento 1.1, exponencial = 0.5, revisión de consulta = 700

4.3.1.11. Experimento 2.1 50 TI/50 TS (700)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.



	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,747	0,000352	0,000123	0,000044
síncrono	0,747	0,000359	0,000174	0,000088
total	4,47	0,002133	0,000901	0,000396

Ilustración 32: experimento 2.1, exponencial = 0.5, revisión de consulta = 700

4.3.1.12. Experimento 3.1 1 TI/1 TS (700)

- Consultas Time-Interval: 347.
- Consultas Time-Slice: 653.

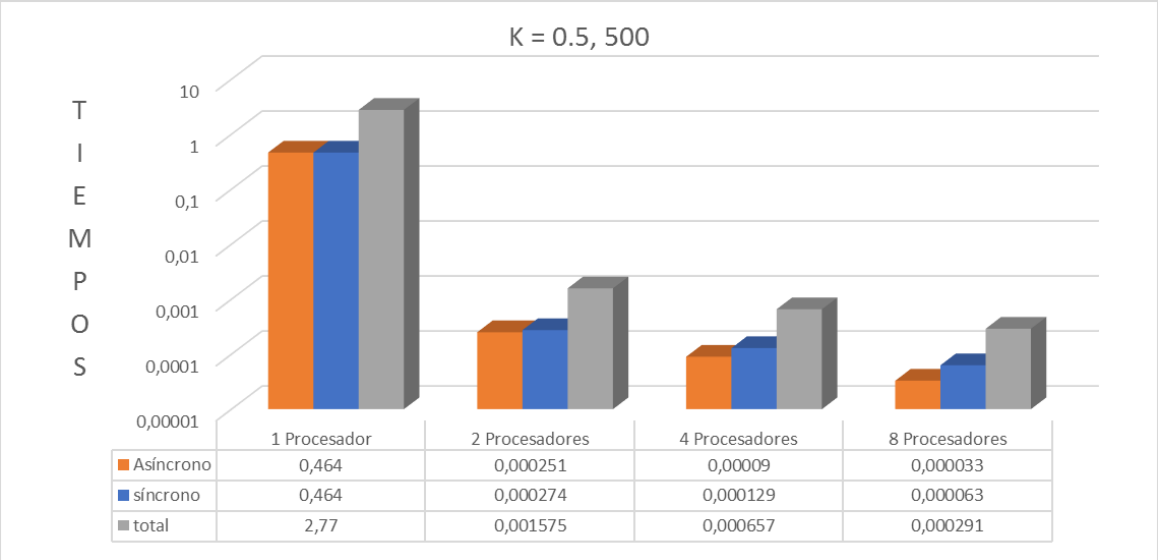
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,644	0,000394	0,000123	0,000042
síncrono	0,644	0,000408	0,000175	0,000088
total	3,91	0,002406	0,000905	0,00039

Ilustración 33: experimento 3.1, exponencial = 0.5, revisión de consulta = 700

A continuación, se cambiará la revisión del algoritmo a la consulta 500, quedando los tiempos de comunicación de la siguiente manera:

4.3.1.13. Experimento 1.1 50 Ts/50 Ti (500)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

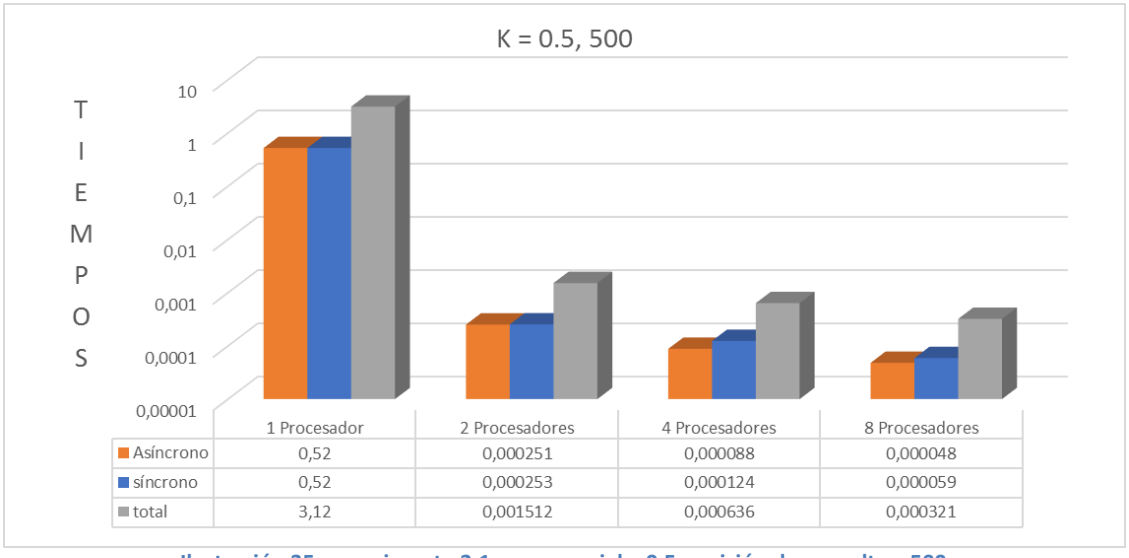


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,464	0,000251	0,00009	0,000033
síncrono	0,464	0,000274	0,000129	0,000063
total	2,77	0,001575	0,000657	0,000291

Ilustración 34: experimento 1.1, exponencial = 0.5, revisión de consulta = 500

4.3.1.14. Experimento 2.1 50 Ti/50 Ts (500)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

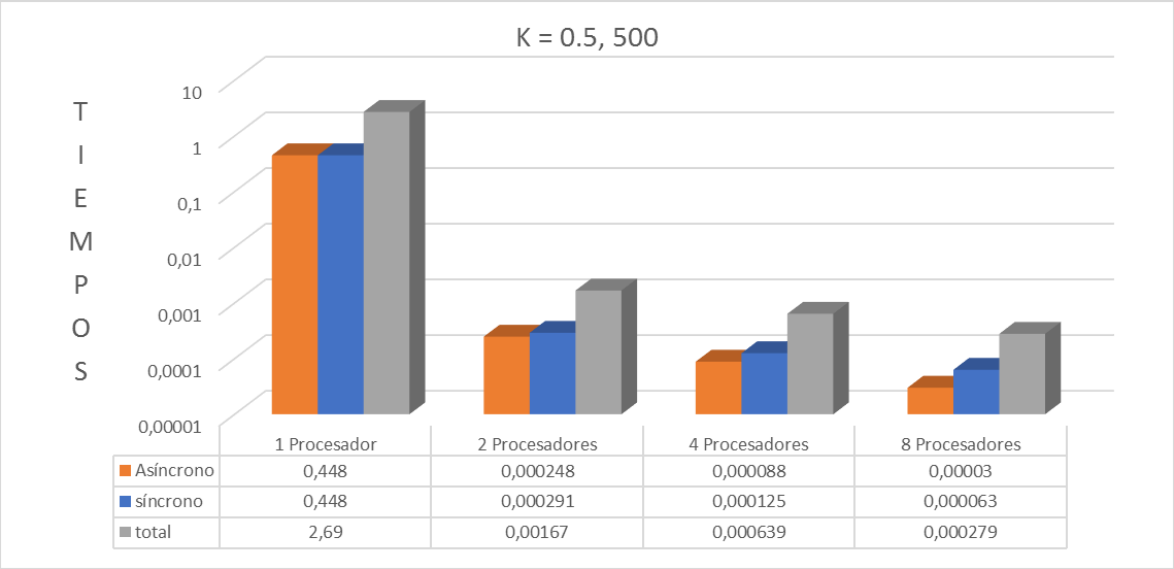


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,52	0,000251	0,000088	0,000048
síncrono	0,52	0,000253	0,000124	0,000059
total	3,12	0,001512	0,000636	0,000321

Ilustración 35: experimento 2.1, exponencial = 0.5, revisión de consulta = 500

4.3.1.15. Experimento 3.1 1 Ti/1 Ts (500)

- Consultas Time-Interval: 347.
- Consultas Time-Slice: 653.



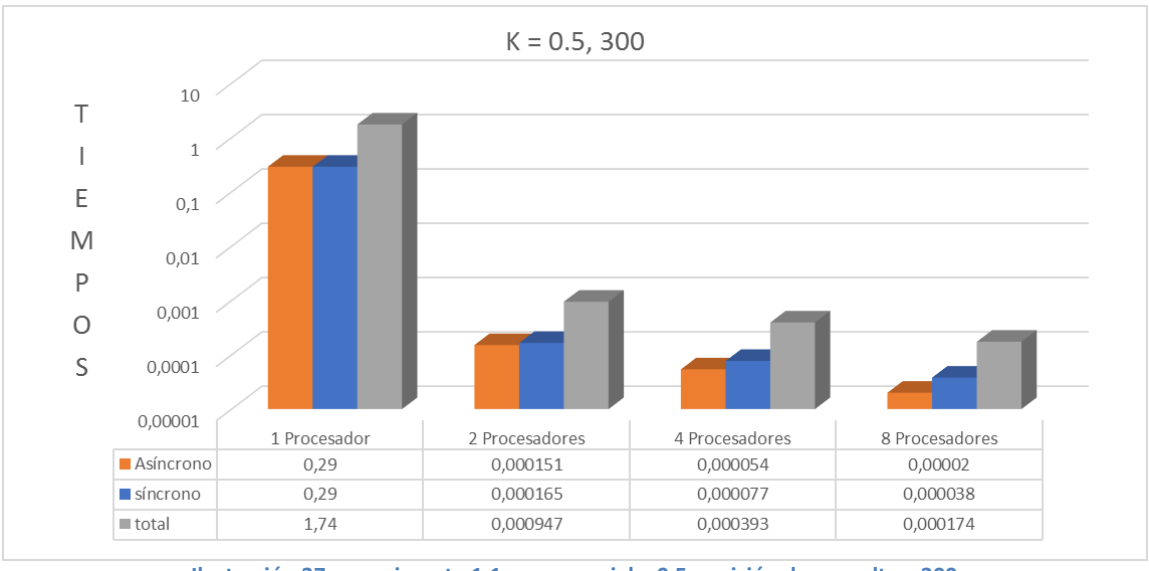
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,448	0,000248	0,000088	0,00003
síncrono	0,448	0,000291	0,000125	0,000063
total	2,69	0,00167	0,000639	0,000279

Ilustración 36: experimento 3.1, exponencial = 0.5, revisión de consulta = 500

A continuación, se cambiará la revisión del algoritmo a la consulta 300, quedando los tiempos de comunicación de la siguiente manera:

4.3.1.16. Experimento 1.1 50 Ts/50 Ti (300)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

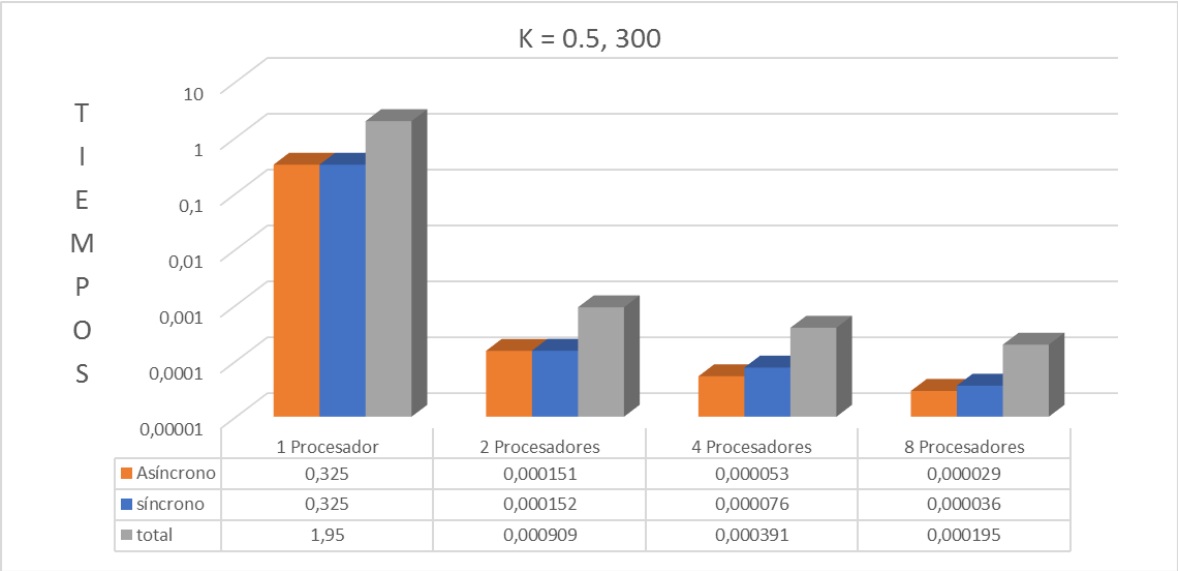


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,29	0,000151	0,000054	0,00002
síncrono	0,29	0,000165	0,000077	0,000038
total	1,74	0,000947	0,000393	0,000174

Ilustración 37: experimento 1.1, exponencial = 0.5, revisión de consulta = 300

4.3.1.17. Experimento 2.1 50 Ti/50 Ts (300)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

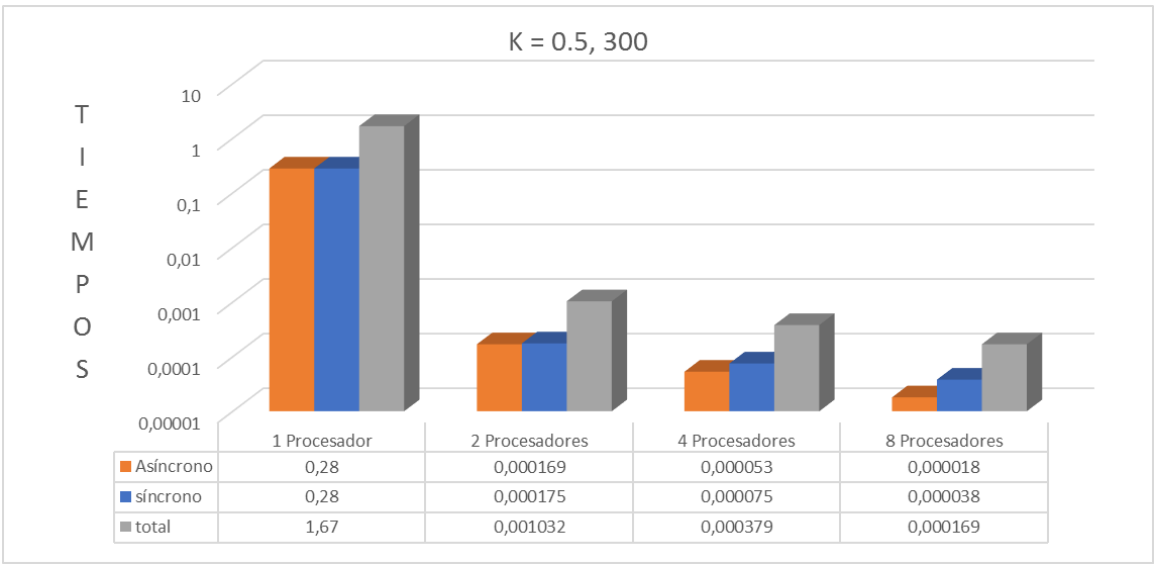


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,325	0,000151	0,000053	0,000029
síncrono	0,325	0,000152	0,000076	0,000036
total	1,95	0,000909	0,000391	0,000195

Ilustración 38: experimento 2.1, exponencial = 0.5, revisión de consulta = 300

4.3.1.18. Experimento 3.1 1 Ti/1 Ts (300)

- Consultas Time-Interval: 347.
- Consultas Time-Slice: 653.



	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
Asíncrono	0,28	0,000169	0,000053	0,000018
síncrono	0,28	0,000175	0,000075	0,000038
total	1,67	0,001032	0,000379	0,000169

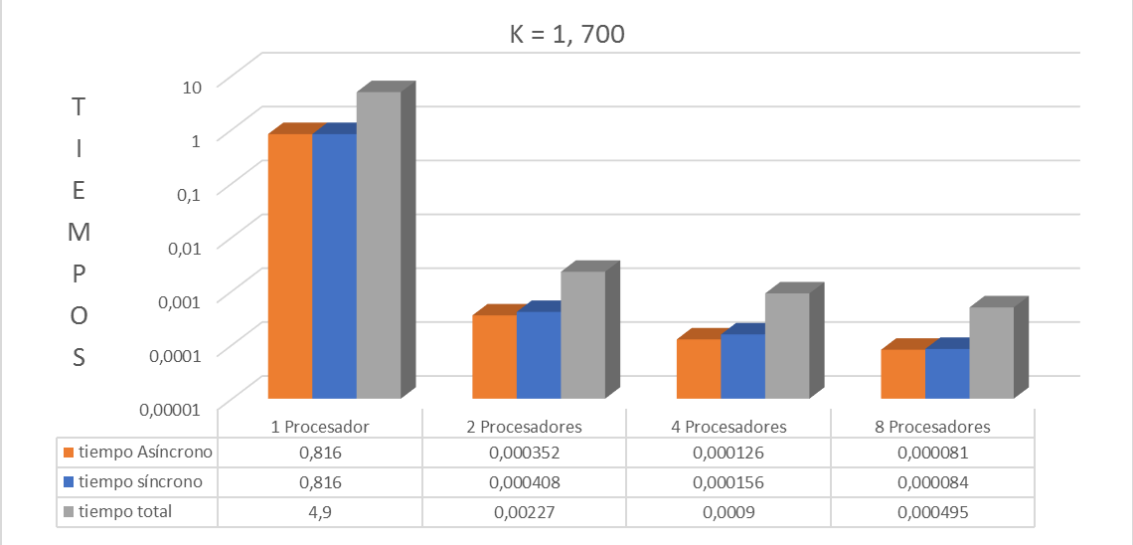
Ilustración 39: experimento 3.1, exponencial = 0.5, revisión de consulta = 300

4.3.2. Distribución Gamma

- Parámetro: 1.
- Revisar cambio de índice: a la consulta 700.
- Movilidad objetos: 10%.

4.3.2.1. Experimento 1.2 50 TS/50 (700)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

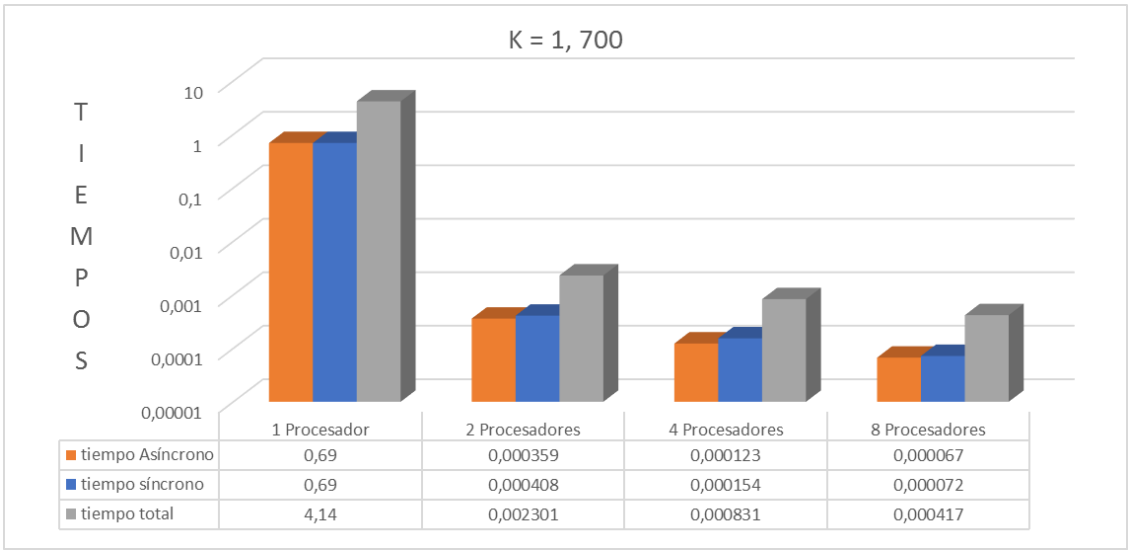


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,816	0,000352	0,000126	0,000081
tiempo síncrono	0,816	0,000408	0,000156	0,000084
tiempo total	4,9	0,00227	0,0009	0,000495

Ilustración 40: experimento 1.2, gamma = 1, revisión de consulta = 700

4.3.2.2. Experimento 2.2 50 TI/50 TS (700)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

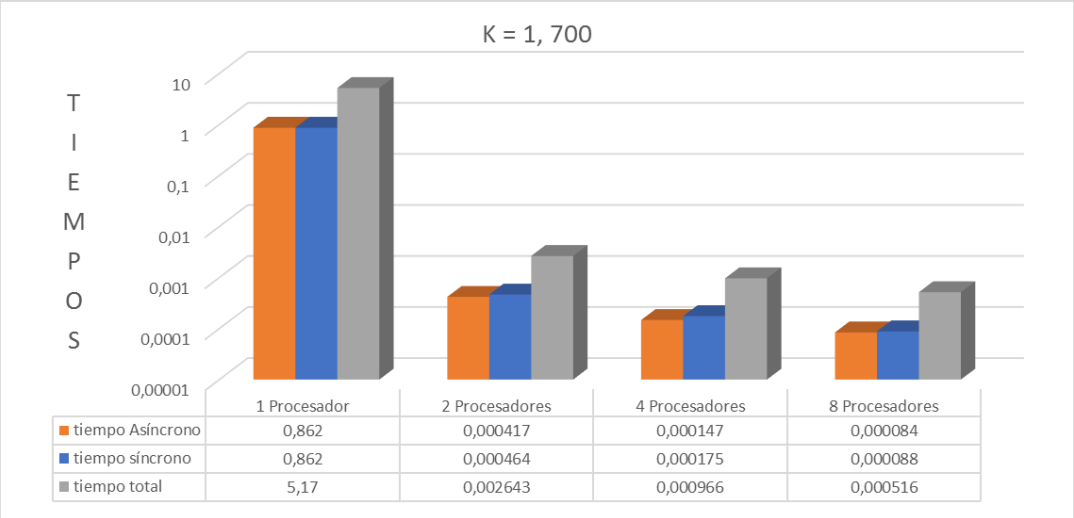


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,69	0,000359	0,000123	0,000067
tiempo síncrono	0,69	0,000408	0,000154	0,000072
tiempo total	4,14	0,002301	0,000831	0,000417

Ilustración 41: experimento 2.2, gamma = 1, revisión de consulta = 700

4.3.2.3. Experimento 3.2 1 TI/1 TS (700)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



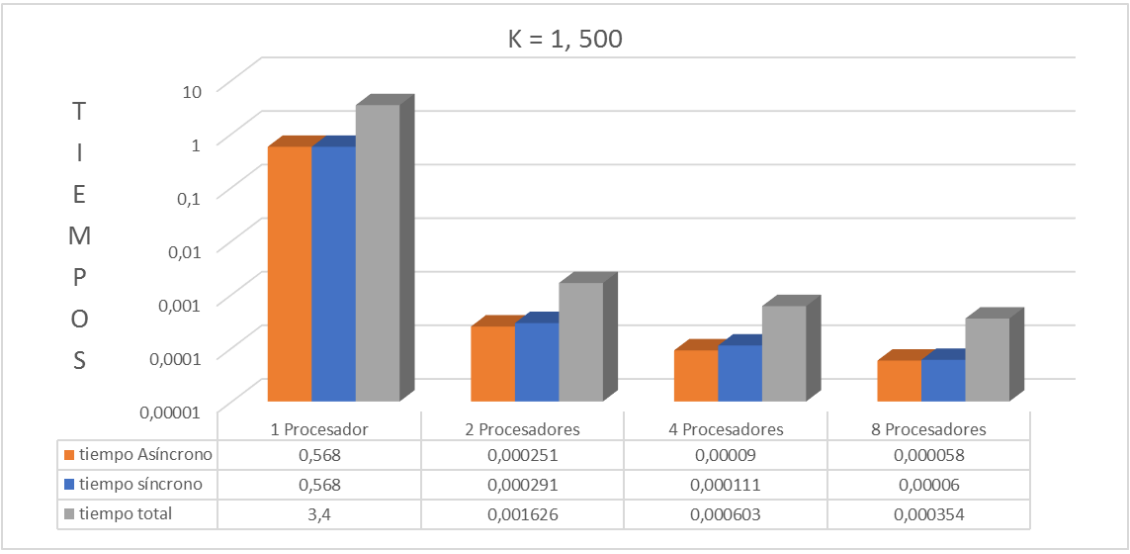
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,862	0,000417	0,000147	0,000084
tiempo síncrono	0,862	0,000464	0,000175	0,000088
tiempo total	5,17	0,002643	0,000966	0,000516

Ilustración 42: experimento 3.2, gamma= 1, revisión de consulta = 700

A continuación, se cambiará la revisión del algoritmo a la consulta 500, quedando los tiempos de comunicación de la siguiente manera:

4.3.2.4. Experimento 1.2 50 Ts/50 Ti (500)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

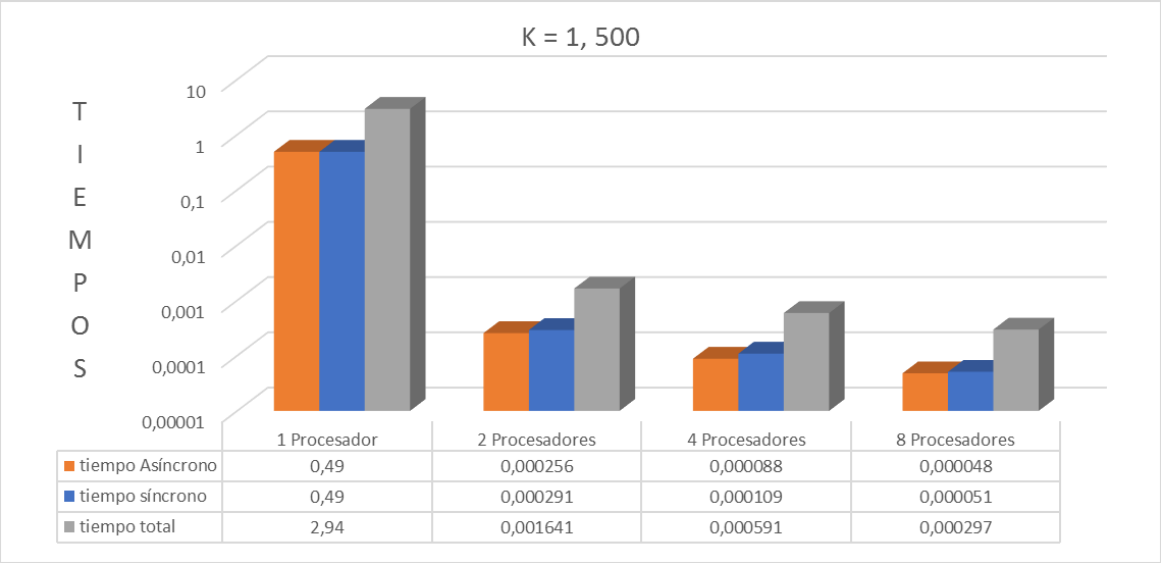


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,568	0,000251	0,00009	0,000058
tiempo síncrono	0,568	0,000291	0,000111	0,00006
tiempo total	3,4	0,001626	0,000603	0,000354

Ilustración 43: experimento 1.2, gamma = 1, revisión de consulta = 500

4.3.2.5. Experimento 2.2 50 Ti/50 Ts (500)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

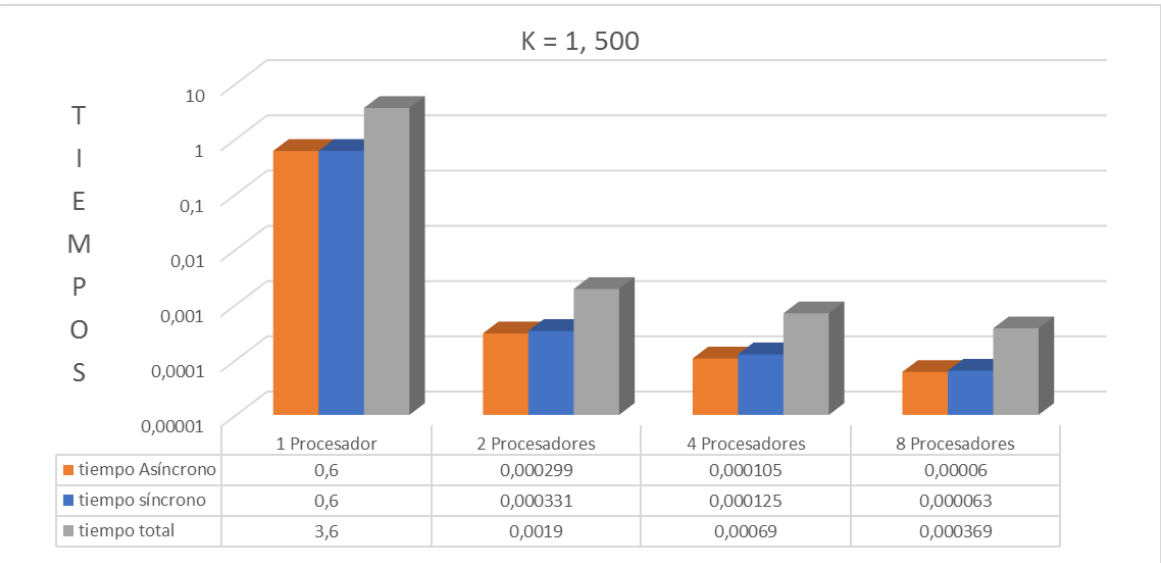


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,49	0,000256	0,000088	0,000048
tiempo síncrono	0,49	0,000291	0,000109	0,000051
tiempo total	2,94	0,001641	0,000591	0,000297

Ilustración 44: experimento 2.2, gamma= 1, revisión de consulta = 500

4.3.2.6. Experimento 3.2 1 Ti/1 Ts (500)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



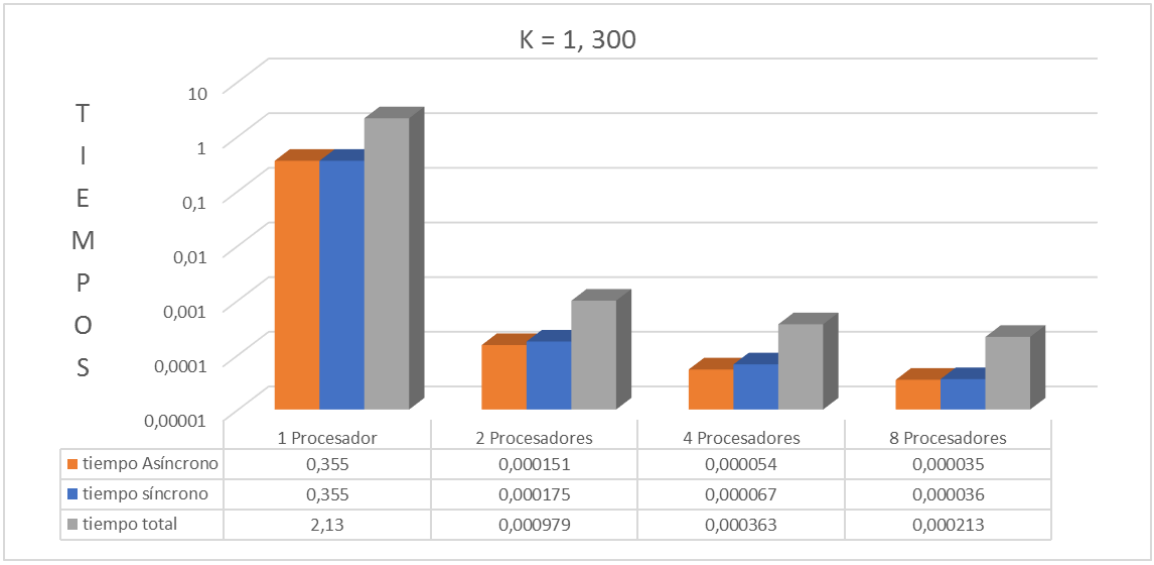
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,6	0,000299	0,000105	0,00006
tiempo síncrono	0,6	0,000331	0,000125	0,000063
tiempo total	3,6	0,0019	0,00069	0,000369

Ilustración 45: experimento 3.2, gamma = 1, revisión de consulta = 500

A continuación, se cambiará la revisión del algoritmo a la consulta 300, quedando los tiempos de comunicación de la siguiente manera:

4.3.2.7. Experimento 1.2 50 Ts/50 Ti (300)

- Consultas Time-Interval: 0.
- Consultas Time-Slice: 1000.

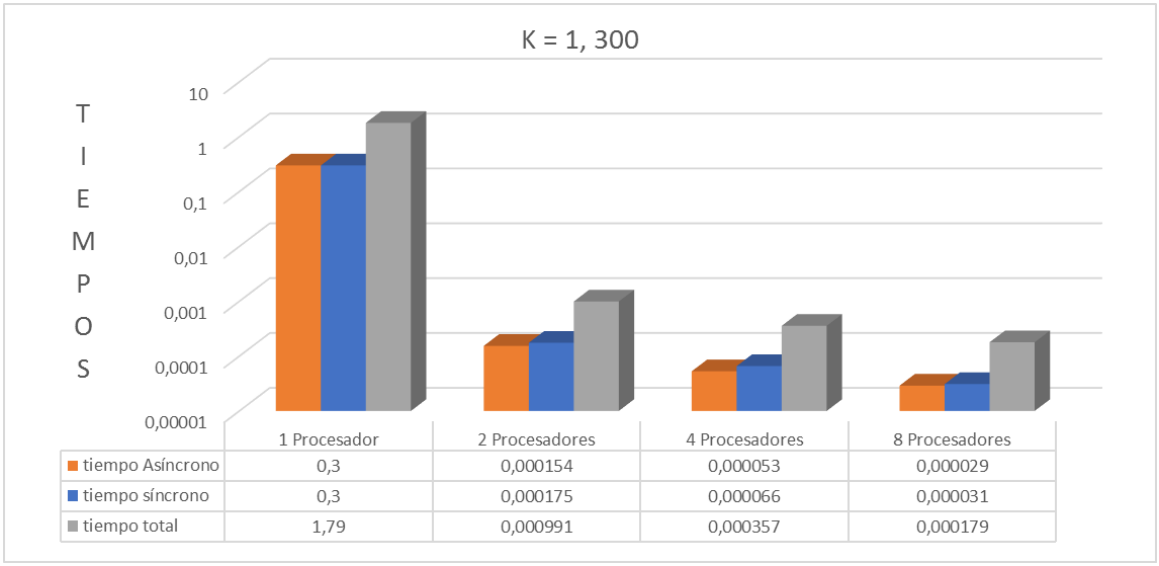


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,355	0,000151	0,000054	0,000035
tiempo síncrono	0,355	0,000175	0,000067	0,000036
tiempo total	2,13	0,000979	0,000363	0,000213

Ilustración 46: experimento 1.2, gamma = 1, revisión de consulta = 300

4.3.2.8. Experimento 2.2 50 Ti/50 Ts (300)

- Consultas Time-Interval: 1000.
- Consultas Time-Slice: 0.

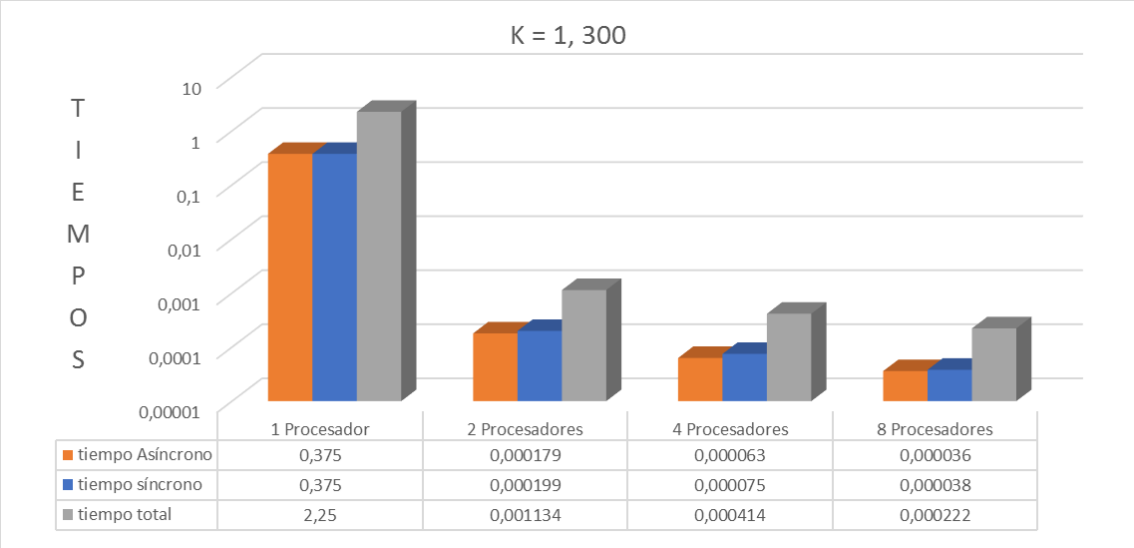


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,3	0,000154	0,000053	0,000029
tiempo síncrono	0,3	0,000175	0,000066	0,000031
tiempo total	1,79	0,000991	0,000357	0,000179

Ilustración 47: experimento 2.2, gamma = 1, revisión de consulta = 300

4.3.2.9. Experimento 3.2 1 Ti/1 Ts (300)

- Consultas Time-Interval: 440.
- Consultas Time-Slice: 560.



	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,375	0,000179	0,000063	0,000036
tiempo síncrono	0,375	0,000199	0,000075	0,000038
tiempo total	2,25	0,001134	0,000414	0,000222

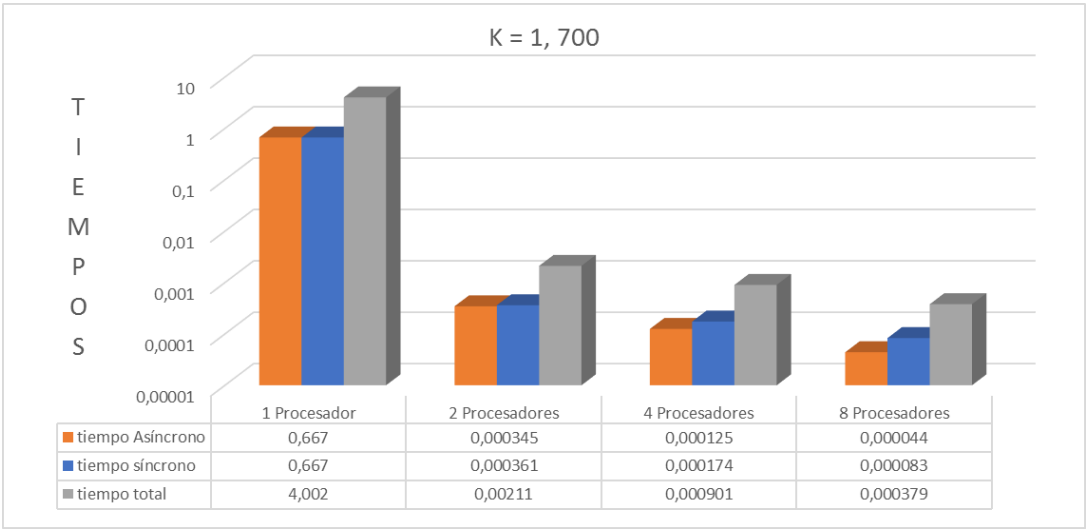
Ilustración 48: experimento 3.2, gamma= 1, revisión de consulta = 300

4.3.3. Distribución Zipf

- Parámetro: 1.
- Revisar cambio de índice: a la consulta 700.
- Movilidad objetos: 10%.

4.3.3.1. Experimento 1.3 50 TS/50 (700)

- Consultas Time-Interval: 115.
- Consultas Time-Slice: 885.

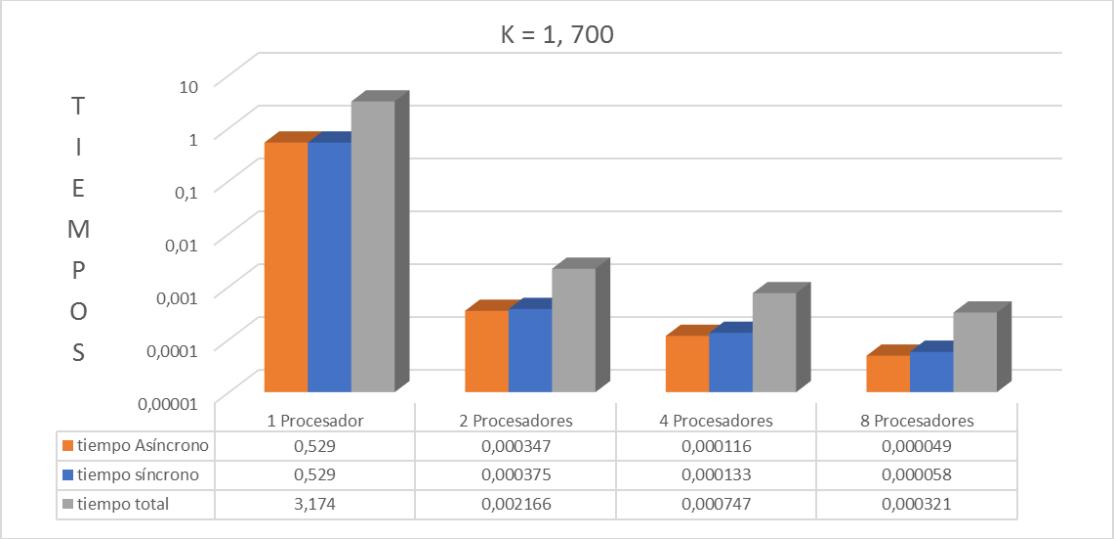


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,667	0,000345	0,000125	0,000044
tiempo síncrono	0,667	0,000361	0,000174	0,000083
tiempo total	4,002	0,00211	0,000901	0,000379

Ilustración 49: experimento 1.1, exponencial = 1, revisión de consulta = 700

4.3.3.2. Experimento 2.3 50 TI/50 TS (700)

- Consultas Time-Interval: 885.
- Consultas Time-Slice: 115.

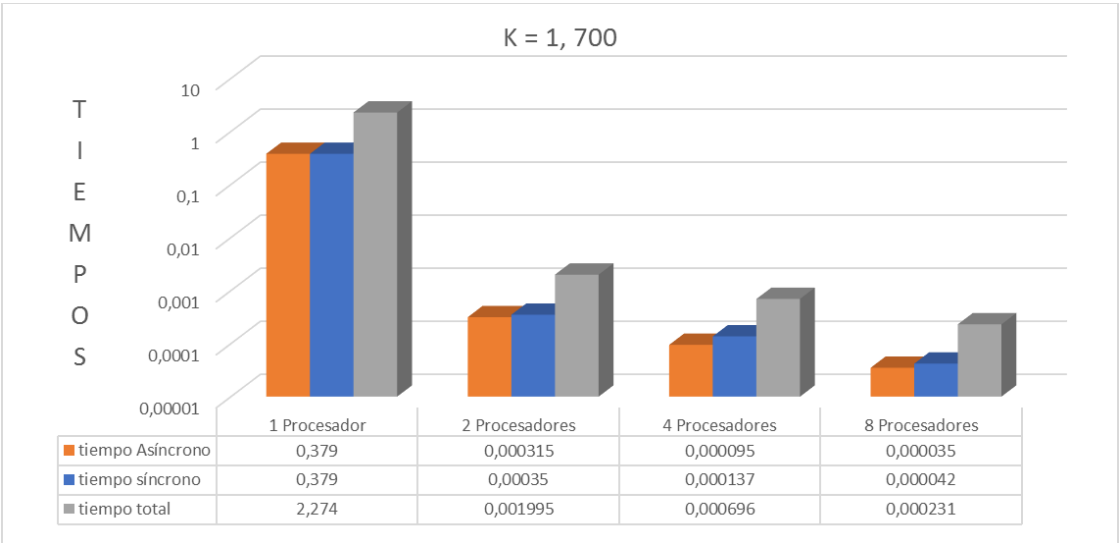


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,529	0,000347	0,000116	0,000049
tiempo síncrono	0,529	0,000375	0,000133	0,000058
tiempo total	3,174	0,002166	0,000747	0,000321

Ilustración 50: experimento 2.3, zipf = 1, revisión de consulta = 700

4.3.3.3. Experimento 3.3 1 TI/1 TS (700)

- Consultas Time-Interval: 440.
- Consultas Time-Slice: 560.



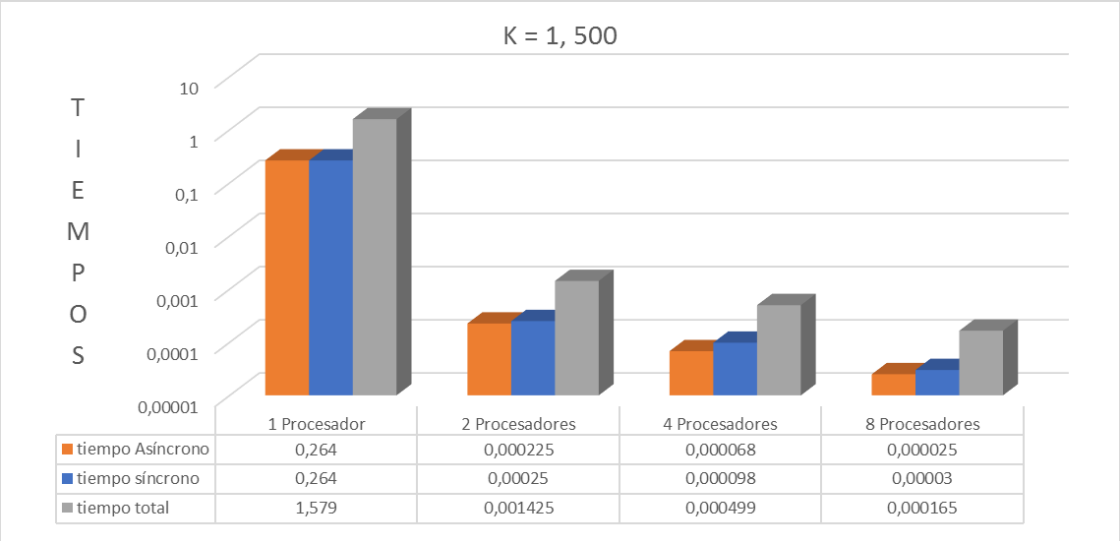
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,379	0,000315	0,000095	0,000035
tiempo síncrono	0,379	0,00035	0,000137	0,000042
tiempo total	2,274	0,001995	0,000696	0,000231

Ilustración 51: experimento 3.3, zipf = 1, revisión de consulta = 700

A continuación, se cambiará la revisión del algoritmo a la consulta 500, quedando los tiempos de comunicación de la siguiente manera:

4.3.3.4. Experimento 1.3 50 Ts/50 Ti (500)

- Consultas Time-Interval: 115.
- Consultas Time-Slice: 885.

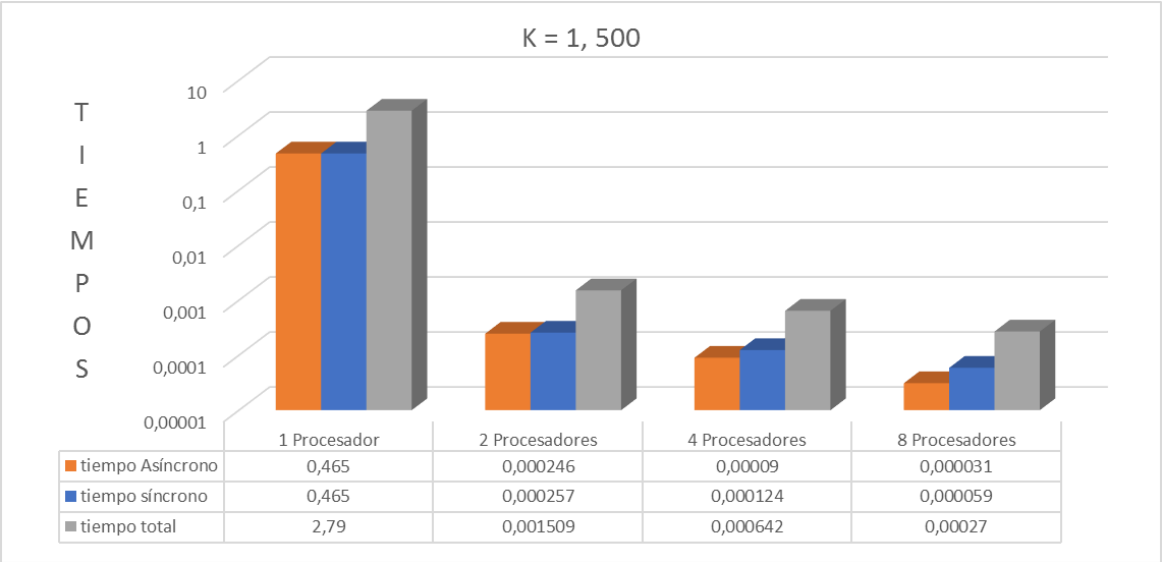


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,264	0,000225	0,000068	0,000025
tiempo síncrono	0,264	0,00025	0,000098	0,00003
tiempo total	1,579	0,001425	0,000499	0,000165

Ilustración 52: experimento 1.3, zipf = 1, revisión de consulta = 500

4.3.3.5. Experimento 2.3 50 Ti/50 Ts (500)

- Consultas Time-Interval: 885.
- Consultas Time-Slice: 115.

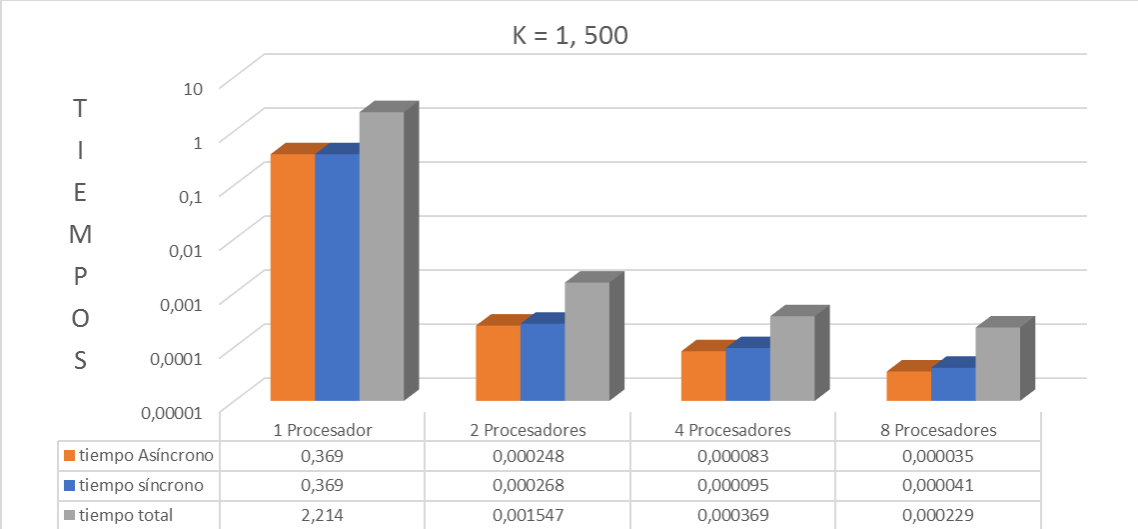


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,465	0,000246	0,00009	0,000031
tiempo síncrono	0,465	0,000257	0,000124	0,000059
tiempo total	2,79	0,001509	0,000642	0,00027

Ilustración 53: experimento 2.3, zipf = 1, revisión de consulta = 500

4.3.3.6. Experimento 3.3 1 Ti/1 Ts (500)

- Consultas Time-Interval: 440.
- Consultas Time-Slice: 560.



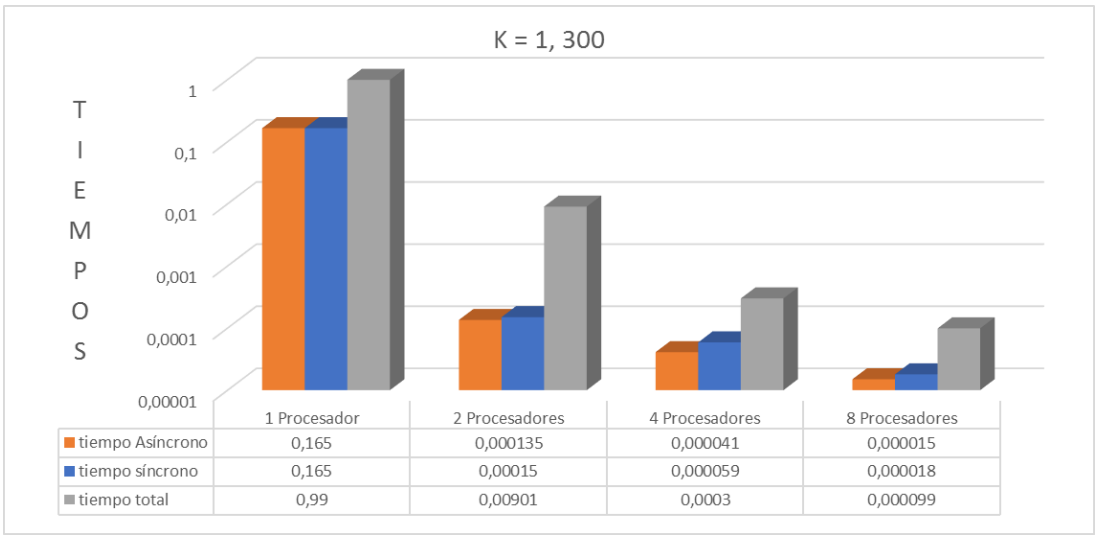
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,369	0,000248	0,000083	0,000035
tiempo síncrono	0,369	0,000268	0,000095	0,000041
tiempo total	2,214	0,001547	0,000369	0,000229

Ilustración 54: experimento 3.3, zipf = 1, revisión de consulta = 500

A continuación, se cambiará la revisión del algoritmo a la consulta 300, quedando los tiempos de comunicación de la siguiente manera:

4.3.3.7. Experimento 1.3 50 Ts/50 Ti (300)

- Consultas Time-Interval: 115.
- Consultas Time-Slice: 885.

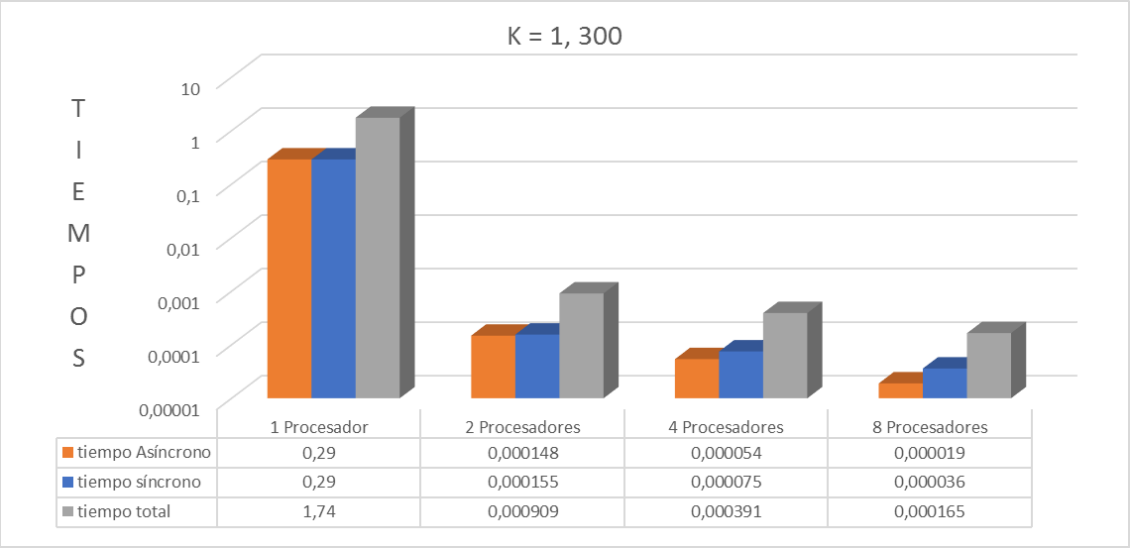


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,165	0,000135	0,000041	0,000015
tiempo síncrono	0,165	0,00015	0,000059	0,000018
tiempo total	0,99	0,00901	0,0003	0,000099

Ilustración 55: experimento 1.3, zipf = 1, revisión de consulta = 300

4.3.3.8. Experimento 2.3 50 Ti/50 Ts (300)

- Consultas Time-Interval: 885.
- Consultas Time-Slice: 115.

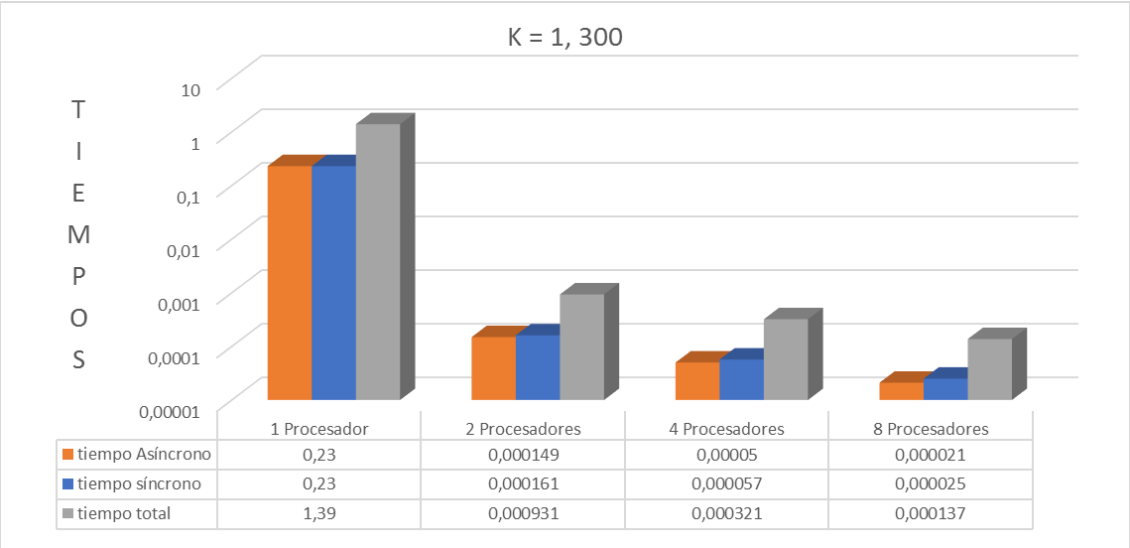


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,29	0,000148	0,000054	0,000019
tiempo síncrono	0,29	0,000155	0,000075	0,000036
tiempo total	1,74	0,000909	0,000391	0,000165

Ilustración 56: experimento 2.3, zipf = 1, revisión de consulta = 300

4.3.3.9. Experimento 3.3 1 Ti/1 Ts (300)

- Consultas Time-Interval: 440.
- Consultas Time-Slice: 560.



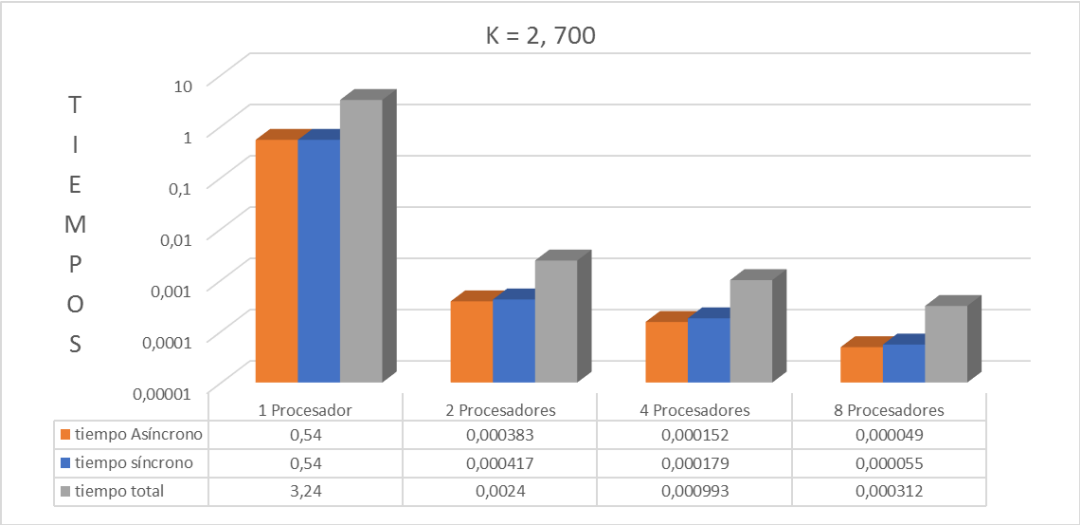
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,23	0,000149	0,00005	0,000021
tiempo síncrono	0,23	0,000161	0,000057	0,000025
tiempo total	1,39	0,000931	0,000321	0,000137

Ilustración 57: experimento 2.3, zipf = 1, revisión de consulta = 300

- Parámetro: 2.
- Revisar cambio de índice: a la consulta 700.
- Movilidad objetos: 10%.

4.3.3.10. Experimento 1.3 50 TS/50 (700)

- Consultas Time-Interval: 4.
- Consultas Time-Slice: 996.

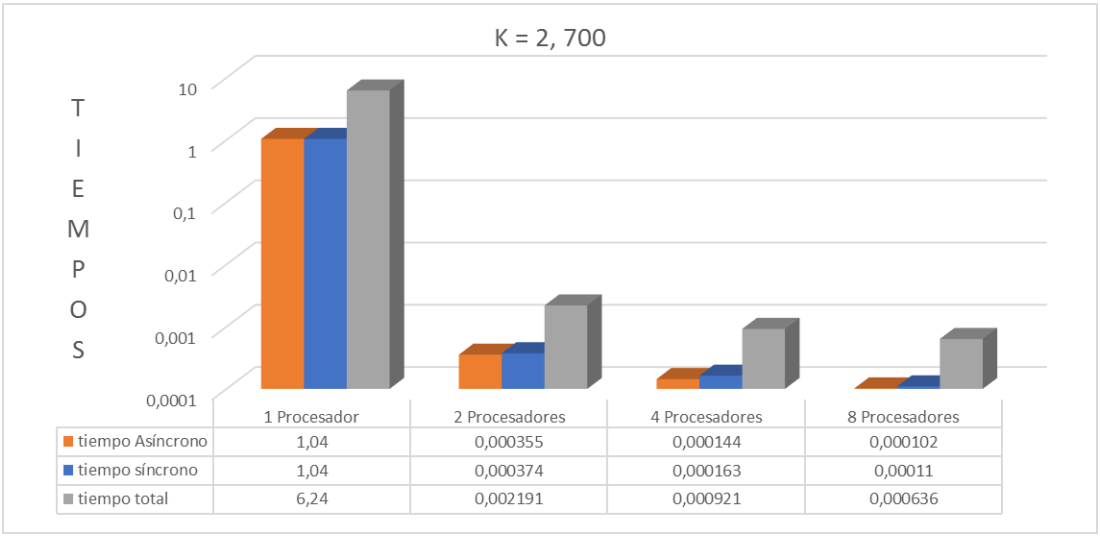


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,54	0,000383	0,000152	0,000049
tiempo síncrono	0,54	0,000417	0,000179	0,000055
tiempo total	3,24	0,0024	0,000993	0,000312

Ilustración 58: experimento 1.3, zipf = 2, revisión de consulta = 700

4.3.3.11. Experimento 2.3 50 TI/50 TS (700)

- Consultas Time-Interval: 996.
- Consultas Time-Slice: 4.

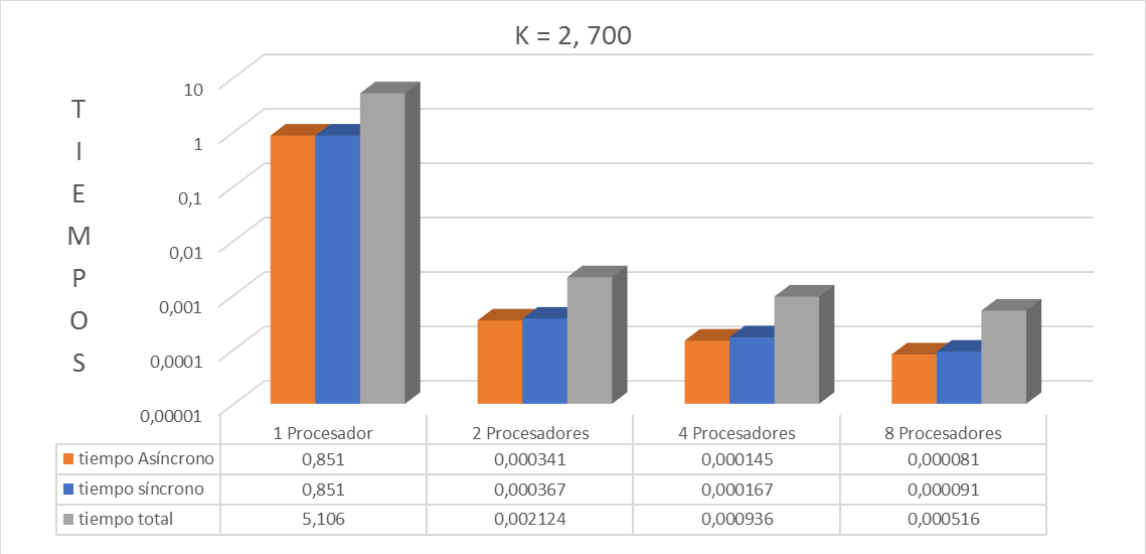


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	1,04	0,000355	0,000144	0,000102
tiempo síncrono	1,04	0,000374	0,000163	0,00011
tiempo total	6,24	0,002191	0,000921	0,000636

Ilustración 59: experimento 2.3, zipf = 2, revisión de consulta = 700

4.3.3.12. Experimento 3.3 1 TI/1 TS (700)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



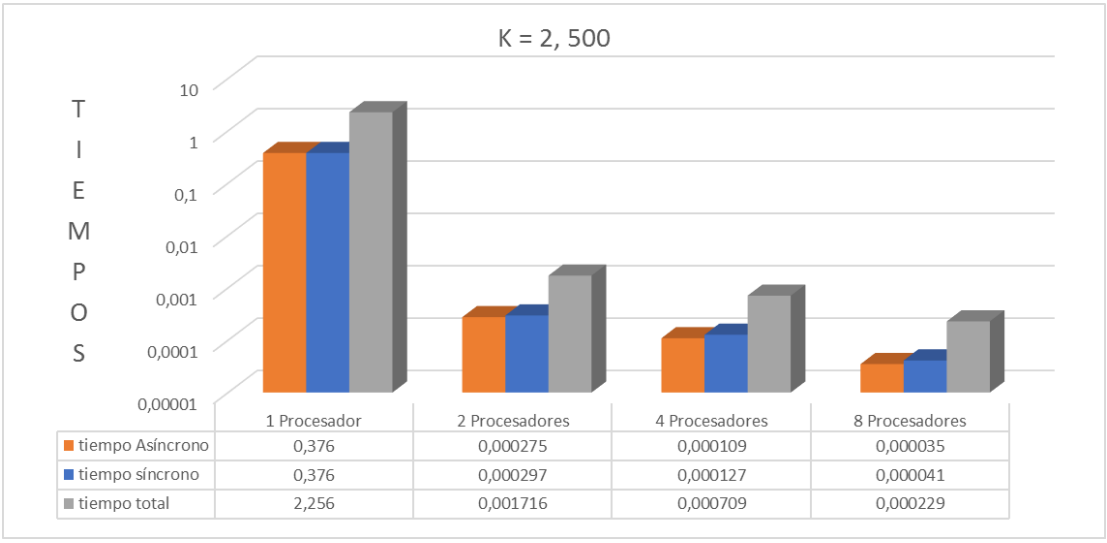
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,851	0,000341	0,000145	0,000081
tiempo síncrono	0,851	0,000367	0,000167	0,000091
tiempo total	5,106	0,002124	0,000936	0,000516

Ilustración 60: experimento 3.3, zipf = 2, revisión de consulta = 700

A continuación, se cambiará la revisión del algoritmo a la consulta 500, quedando los tiempos de comunicación de la siguiente manera:

4.3.3.13. Experimento 1.3 50 Ts/50 Ti (500)

- Consultas Time-Interval: 4.
- Consultas Time-Slice: 966.

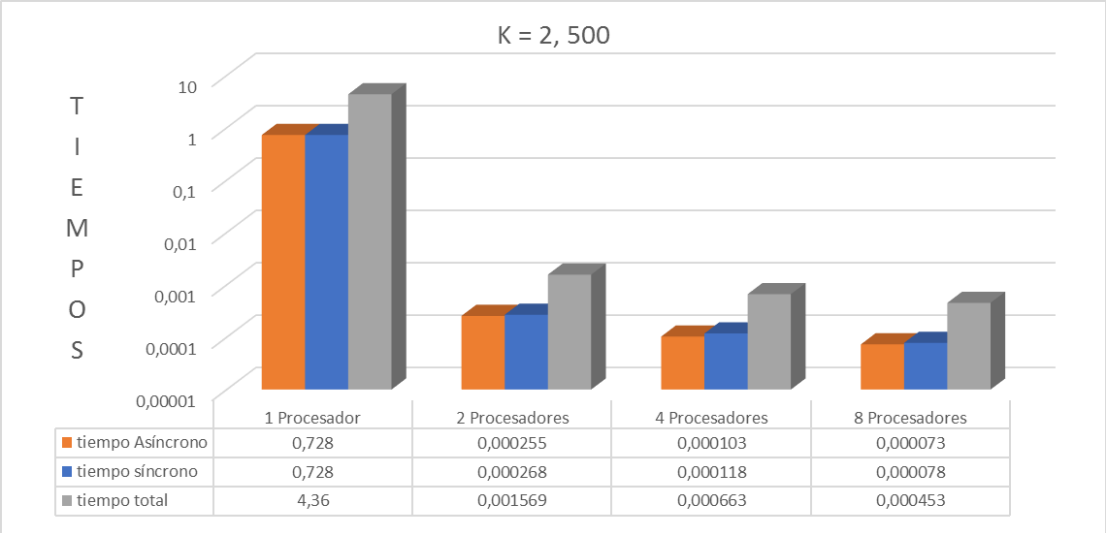


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,376	0,000275	0,000109	0,000035
tiempo síncrono	0,376	0,000297	0,000127	0,000041
tiempo total	2,256	0,001716	0,000709	0,000229

Ilustración 61: experimento 1.3, zipf = 2, revisión de consulta = 500

4.3.3.14. Experimento 2.3 50 Ti/50 Ts (500)

- Consultas Time-Interval: 996.
- Consultas Time-Slice: 4.

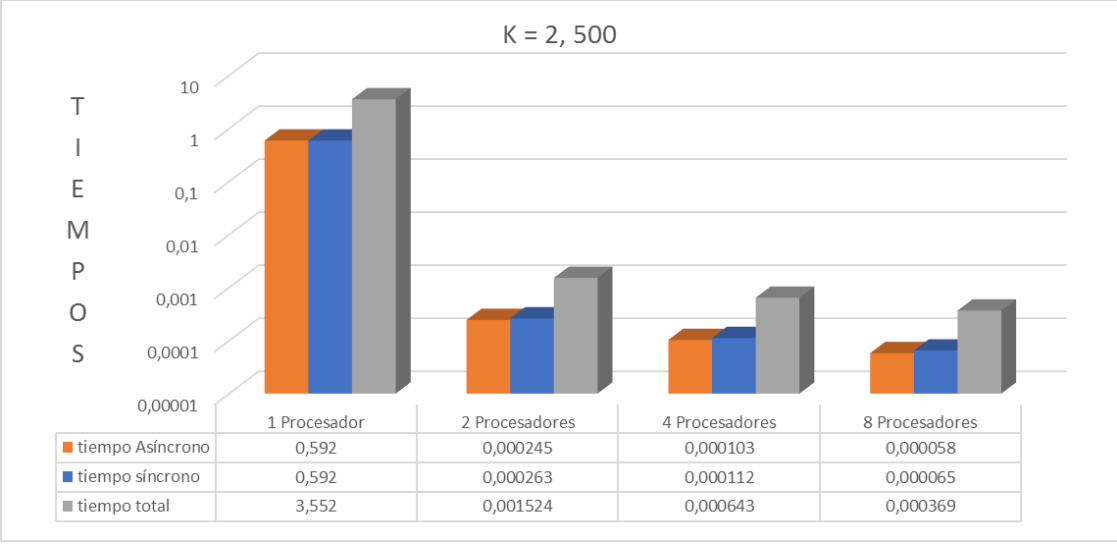


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,728	0,000255	0,000103	0,000073
tiempo síncrono	0,728	0,000268	0,000118	0,000078
tiempo total	4,36	0,001569	0,000663	0,000453

Ilustración 62: experimento 2.3, zipf = 2, revisión de consulta = 500

4.3.3.15. Experimento 3.3 1 Ti/1 Ts (500)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



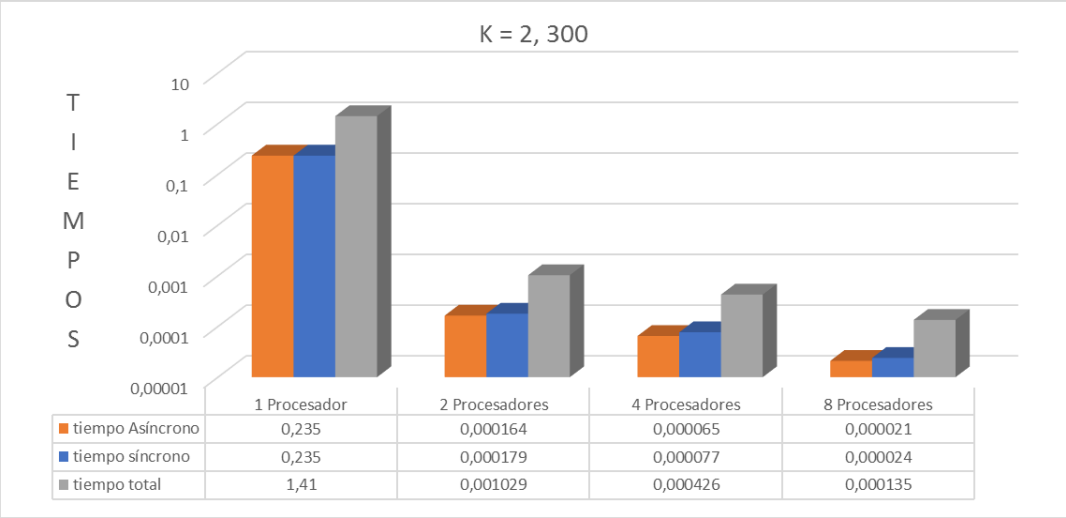
	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,592	0,000245	0,000103	0,000058
tiempo síncrono	0,592	0,000263	0,000112	0,000065
tiempo total	3,552	0,001524	0,000643	0,000369

Ilustración 63: experimento 3.3, zipf = 2, revisión de consulta = 500

A continuación, se cambiará la revisión del algoritmo a la consulta 300, quedando los tiempos de comunicación de la siguiente manera:

4.3.3.16. Experimento 1.3 50 Ts/50 Ti (300)

- Consultas Time-Interval: 4.
- Consultas Time-Slice: 996.

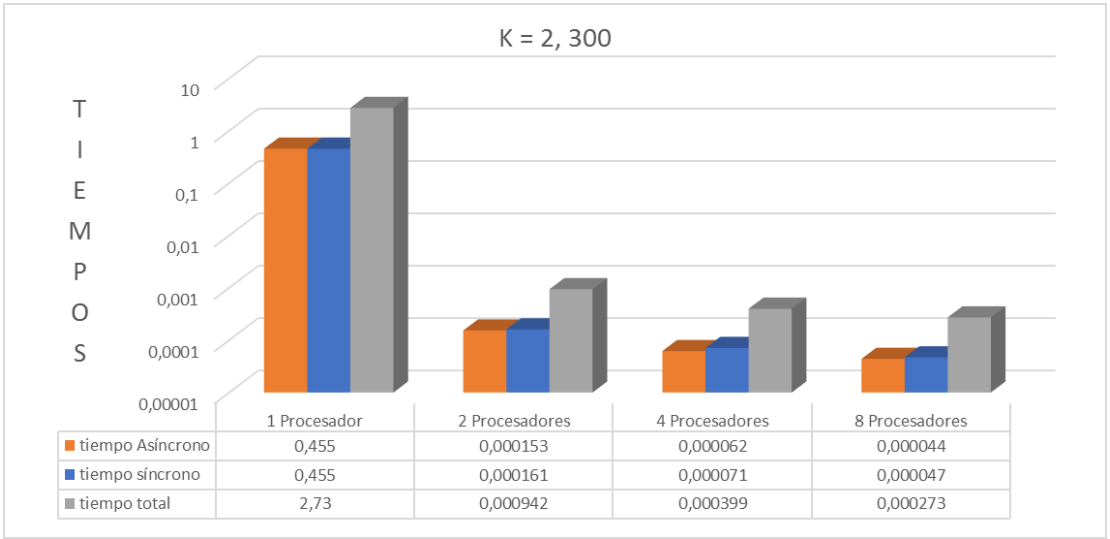


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,235	0,000164	0,000065	0,000021
tiempo síncrono	0,235	0,000179	0,000077	0,000024
tiempo total	1,41	0,001029	0,000426	0,000135

Ilustración 64: experimento 1.3, zipf = 2, revisión de consulta = 300

4.3.3.17. Experimento 2.3 50 Ti/50 Ts (300)

- Consultas Time-Interval: 996.
- Consultas Time-Slice: 4.

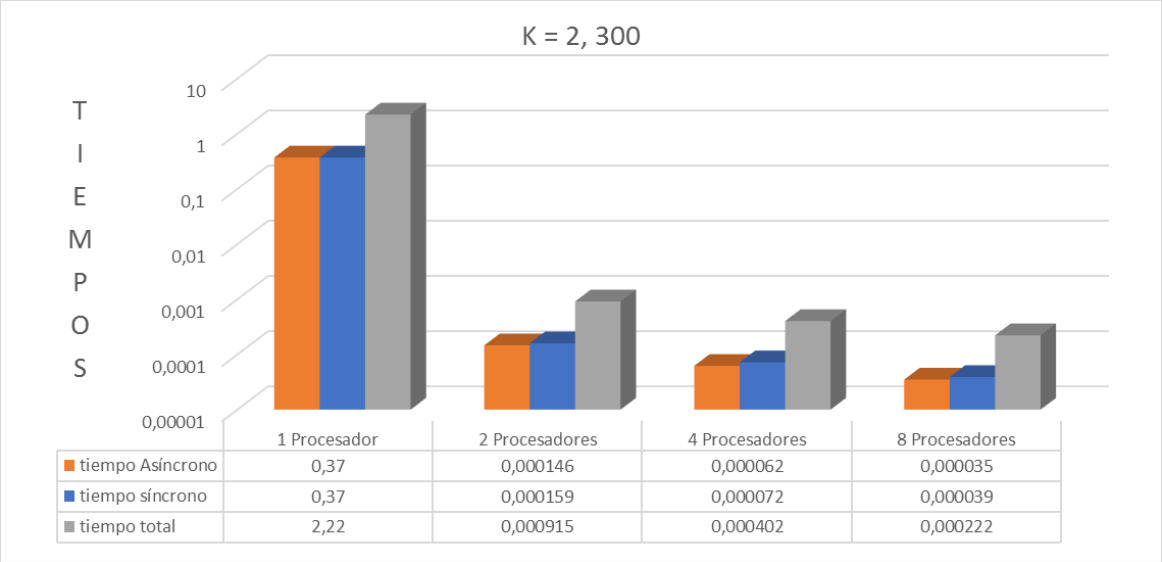


	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,455	0,000153	0,000062	0,000044
tiempo síncrono	0,455	0,000161	0,000071	0,000047
tiempo total	2,73	0,000942	0,000399	0,000273

Ilustración 65: experimento 2.3, zipf = 2, revisión de consulta = 300

4.3.3.18. Experimento 3.3 1 Ti/1 Ts (300)

- Consultas Time-Interval: 260.
- Consultas Time-Slice: 740.



	1 Procesador	2 Procesadores	4 Procesadores	8 Procesadores
tiempo Asíncrono	0,37	0,000146	0,000062	0,000035
tiempo síncrono	0,37	0,000159	0,000072	0,000039
tiempo total	2,22	0,000915	0,000402	0,000222

Ilustración 66: experimento 3.3, zipf = 2, revisión de consulta = 300

Capítulo V

5. Conclusiones

Objetivo General

Analizar, diseñar e implementar diversas estrategias de comunicación para obtener los costos comunicación en un sistema distribuido, utilizando un algoritmo en línea para el balance de carga en una Base de Datos Espacio-Temporales. Para ello se utilizarán los enfoques (algoritmos) de tres proyectos de título anteriores, con el propósito analizar, diseñar, implementar y obtener resultados experimentales de los costos de comunicación de dichos algoritmos en un ambiente distribuido.

Se logró luego de analizar, modificar y ejecutar varias pruebas en el algoritmo utilizado en los proyectos de títulos anteriores, poder obtener el tiempo de comunicación asociado a los diferentes experimentos realizados con diferentes distribuciones probabilísticas, Zipf-Gamma-Exponencial, así como a un número determinado de consultas 300, 500 y 700, se utilizaron estrategias tanto sincrónica como asincrónica, tiendo como resultado que a mayor cantidad de procesadores el tiempo de comunicación tiene un menor costo que en menos procesadores y que además las diferencias entre los tiempos asincrónicos y sincrónicos es mínima, y que a mayor cantidad de procesadores es más notoria.

Además, se puede concluir que el tiempo se ve influenciado por el tipo de consulta resultando con un costo mayor de comunicación las consultas del tipo time-interval en relación a las consultas time-slice

Objetivos Específicos

- **Estudiar los algoritmos propuestos en los tres proyectos de título anteriores.**

  Gracias al estudio de los algoritmos fue posible llevar a cabo este proyecto, realizar los experimentos y tener mayor conocimiento acerca del tema.

- **Analizar y diseñar diversas estrategias de comunicación en un ambiente distribuido para los algoritmos propuestos en los proyectos de título anteriores.**

  Se estudiaron dos técnicas de comunicación, las cuales fueron sincrónica y asincrónica, en las cuales la comunicación sincrónica hace el envío de datos y luego espera que todos los datos se junten para ser devueltos al nodo principal, mientras que la comunicación asincrónica a medida que va recibiendo datos los va devolviendo en forma inmediata al nodo principal, y teniendo como resultado que por una mínima diferencia el costo de comunicación asincrónica es menor que al de la comunicación sincrónica.

- **Analizar y determinar distintos escenarios de experimentación acorde a la distribución de los objetos en movimiento (esto es sobre el espacio), así como también considerando los tipos de consultas más frecuentes (en este proyecto de título consideraremos sólo 2, time-slice y time-interval).**

  Se crearon tres escenarios de experimentación para el ámbito de consultas y tres escenarios para objetos. Logrando determinar estos escenarios acordes a la distribución de los objetos, ayudados por distribuciones de probabilidad, lo que permitió simular un algoritmo en línea y así, experimentar ampliamente con el algoritmo creado.

Referencias

AHN H., MAMOULIS N. y WONG H. (2001). A survey on multidimensional access methods.

BECKMANN N., KRIEGEL H., SCHNEIDER R., SEEGER B. (1990). The R*-Tree: An efficient and robust access method for points and rectangles. In ACM SIGMOD Conference on Management of Data. ACM, 1990.

CALHOUN V., ADALI T. y PEARLSON G. (1999). Non-stationarity of temporal dynamics in fMRI. In Annual Conference of Engineering in Medicine and Biology, Volume 2, 1999.

CHEN Z., LI C., PEI J., TAO Y., WANG H., WANG W., YANG J., YANG J. y ZHANG D. (2003). Recent progress on selected topics in database research: a report by nine young chinese researchers working in the united states. Journal of Computer Science and Technology, Vol. 18, N° 5, pp. 538-552, 2003.

DEWIT D., GRAY J. (1992). Parallel Database Systems: The future of high performance database processing.

FLYNN, **M.** (1972). Some computer organizations and their effectiveness, IEEE Trans. Comput., Vol. C-21, pp. 948.

FOISY C., CHAILLOUX E. (1995). Caml Flight: a portable SPMD extension of ML for distributed memory multiprocessors.

FOSTER I. (1995). Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering (Paperback), Addison Wesley; 1st edition, 1995.

GAEDE V. y GÜNTHER O. (1998). Multidimensional access methods. ACM computing surveys, Vol. 30, N° 2, pp. 170-231, 1998.

GUTIERREZ G. (2003). Propuesta de un método de acceso espacio-temporal. II WorkShop de Bases de datos. 2003.

GUTIERREZ G., **NAVARRO G., RODRIGUEZ A.** (2006). $Sest_L$: An event-oriented spatio-temporal access method. Technical ReportTR/DCC-2006-5, Department of Computer Science, Universidad de Chile, 2006.

GUTTMAN A. (1984). "R-trees: a dynamic index structure for spatial searching", In Proceedings of ACM SIGMOD Conference on Management of Data, 1984.

HADJIELEFTHERIOU M., KRIAKOV M., TAO Y., KOLLIOS G., DELIS A., TSOTRAS V. (2004). Spatio-Temporal data services in a shared-nothing environment. Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04) - Volume 00, 2004.

JEONG S., NORMAN P., FERNADES A., GRIFFITHS T. (2004). An experimental performance evaluation of spatio-temporal join strategies.

KANG H., CHUNG C. (2002) Exploiting Versions for On-line Data Warehouse Maintenance in MOLAP Servers. 742-753. VLDB 2002, Hong-Kong.

KOLLIOS G., TSOTRAS V., DELIS A. (2000). Indexing problems in spatio-temporal databases.

KUMAR V., GRAMA A., GUPTA A., KARYPIS G., CUMMINGS B. (1994). Introduction to parallel computing: design and analysis of algorithms.

MOORE G. (1965). Cramming more components onto integrated circuits. Electronics, Volume 38, Number 8, 1965.

NASCIMENTO M., SILVA J. (1998). Towards Historical R-Tress. ACM SAC, 1998.

NASCIMENTO M., SILVA J. y THEODORIDIS Y. (1998). Access structures for moving points. Technical Report TR–33, TIME CENTER, 1998.

NASCIMENTO M., SILVA J. y THEODORIDIS Y. (1999). Evaluation of access structures for discretely moving points. Proceedings of the International Workshop on Spatio-Temporal Database Management, pp. 171 – 188, 1999.

PFOSER D. y TRYFONA N. (1998). Requirements, definitions and notations for spatiotemporal application environments. In Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems (GIS'98), pp. 124-130. ACM Press, 1998.}

QUINN M. (2004). Parallel Programming in C with MPI and OpenMP. McGraw Hill, New York, 2004.

REYES V. (2003). Procesamiento Paralelo en Redes Linux Utilizando MPI.

THEODORIDIS Y., SELLIS T., PAPADOPOULUS A. y MANOLOPOULUS Y. (1998). Specifications for efficient indexing in spatiotemporal databases. In IEEE Proceedings of the 10th International Conference on Scientific and Statistical Database Management, pp. 123–132, 1998.

THEODORIDIS Y., STEFANAKIS E., SELLIS T. (2000). Efficient cost models for spatial queries using R-trees"**.** IEEE Transactions on Knowledge and Data Engineering, pp. 19–32, 2000.

THEODORIDIS Y., VAZIRGIANNIS M., SELLIS T. (1996). Spatio-temporal indexing for large multimedia applications. Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems (ICMCS), 1996.

WANG X., ZHOU X. y LU S. (2000). Spatiotemporal data modeling and management: a survey. Proceedings 36th International Conference on Technology of Object-Oriented Languages and System, pp. 202-211, 2000.

WIKIPEDIA. (2007). http://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes.

XU X., HAN J., LU W. (1990) .RT-tree: An improved R- tree index structures for spatio-temporal data. In Proc. of the 4th Intl. Symposium on Spatial Data Handling, pp. 1040 - 1049, 1990.

TAO Y., PAPADIAS D. (2000). MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. Technical Report HKUST-CS00-06, 2000.

ZEILER T.L. (2002). LANDSAT Program Report 2002. Technical report, U.S. Geological Survey - U.S. Department of Interior, Sioux Falls, SD, 2002.

Anexo

Proyecto.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <assert.h>             // Needed for assert() macro
#include <math.h>               // Needed for pow()

//-----  Constants ----------------------------------------
-----------
#define  FALSE           0      // Boolean false
#define  TRUE            1      // Boolean t



//estructuras
struct objeto
{
int  oid;
float ti;
float x1;
float y1;
float x2;
float y2;
int p;
struct objeto
*next; };


//pedir memoria
struct objeto *objeto();


//funciones void
crearSet();
void  leerobjeto();
void estadistica();
void menu();
void  distribucion();
void   distribuir2();
void   distribuir4();
void   distribuir8();
void  distribuir16();
void distribuir1();
int zipf(double alpha, int n); // Returns a Zipf random
variable
```

```
double   rand_val(int seed);          // Jain's RNG


int main(int argc, char **argv)
{

           menu();

           return (0);
}

//reserva de memoria recursivo
struct objeto *newobjeto(){
return((struct objeto *)malloc(sizeof(struct objeto)));
}

//Creación de Archivo Set.
void crearSet(){

      int movimientos,total,totalMoviles,opcion,estaticos=0;
      FILE *ptr;
      float timeStamp=0.000000, var = 0.000000;
      double alpha;                  // Alpha parameter
      double n;                      // N parameter
      int    num_values;             // Number of values
      int    zipf_rv;
       char   temp_string[256];      // Temporary string variable
       int contTS;
       int l,k;


      rand_val(2);
      printf("Ingrese la cantidad de
      objetos\n"); scanf("%d",&total);
      printf("Ingrese la cantidad de
      TimeStamp\n"); scanf("%d",&contTS);

        // Prompt for alpha value
  printf("Alpha value ========================================>
");
  scanf("%s", temp_string);
  alpha = atof(temp_string);

  // Prompt for N value
  printf("N value ============================================>
");
  scanf("%s", temp_string);
  n = atoi(temp_string);

  // Prompt for number of values to generate
  printf("Number of values to generate =======================>
");
```

```
  scanf("%s", temp_string);
  num_values = atoi(temp_string);

      int oid[total];

            do{
                  printf("Ingrese la cantidad de objetos que se
moverán\n");
                  scanf("%d",&totalMoviles);
            }while((totalMoviles%2)!=0);

            int i,r,m;
            float x,y;

            float coordenadas[2][total];
            float coordenadas2[2][totalMoviles];

            ptr = fopen ("set","w");
            for(i=0;i<total;i++) //Creacion primer TimeStamp
            {
                  oid[i]=i+1;
                  x = ((float)rand()/(float)(RAND_MAX)) * 0.800000;
                  y = ((float)rand()/(float)(RAND_MAX)) * 0.800000;
                  coordenadas[0][i] = x;
                  coordenadas[1][i] = y;
                  fprintf(ptr,"%d %f %f %f %f %f
1\n",oid[i],(timeStamp+var),x,y,x,y);

            }
            fclose(ptr);
            int oid2[totalMoviles]; int
            j,bandera=0;
            for(i=0;i<totalMoviles;i++)
            {
                  bandera=0;
                  r = rand() % (total-1);
                  r = r+1;
                  if(i>0) //if para que no se repitan objetos
                  {
                        for(j=0;j<i;j++)
                        {
                              if(r == oid2[j]) bandera=1;

                        }
                  }

                  if (bandera==0){
                              for(l=0;l<total;l++
                              )
                              {
                                    if(oid[l]==r)
                                    {
                                          oid2[i]= r;
```

```
      coordenadas2[0][i]=coordenadas[0][l];

      coordenadas2[1][i]=coordenadas[1][l];
                                    }

                              }

                  }
                  else{
                        i--;
                  }

            }


            ptr = fopen("set","a"); for(k=1;k<contTS;k++)//Creacion
            TimeStamp desde 1 hasta
N
            {
                        var = var + 0.005000;


                  for(i=0;i<totalMoviles;i++)
                  {
                        r = oid2[i];

                        x=coordenadas2[0][i];
                        y=coordenadas2[1][i];
                        zipf_rv = zipf(alpha, n);
                        printf("zipf === %d\n", zipf_rv);
                        if(zipf_rv == 1){
                              x=x+0.01;
                              coordenadas2[0][i]=x;
                        }
                        if(zipf_rv == 2){
                              y=y+0.01;
                              coordenadas2[1][i]=y;
                        }
                        fprintf(ptr,"%d %f %f %f %f %f
1\n",oid2[i],(timeStamp+var),x,y,x,y);

                  }

            }
      fclose(ptr);
      estaticos=total-
      totalMoviles; menu();

}

//leer objetos de archivo Set.
```

```
void leerobjeto(){ char
      buffer[80]; char
      buffer2[80]; char
      buffer3[80]; char
      buffer4[80]; char
      buffer5[80]; char
      buffer6[80]; char
      buffer7[80]; char
      buffer8[80];


      struct objeto
      *r,*p,*q,*headobjeto; int i;
      FILE *ptr;

      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);




      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("set", "r");

      //ver si existe archivo
      set. if(ptr==NULL){
                printf ("\nNo existe el archivo set");
                menu();
      }
      //recorrido de archivo
      set. while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      //obtener linea por linea el contenido del
set.
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
```

```
                        q->x1=atof(buffer4);
                        q->y1=atof(buffer5);
                        q->x2=atof(buffer6);
                        q->y2=atof(buffer7);
                        q->p=atoi(buffer8);



                                    p->next=q;
                                    r=p;
                                    p=q; q-
                                    >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);

            p=headobjeto;
            printf("\n**********LISTA DE OBJETOS************\n");
            //imprimir lista de objetos.
            while(p->next!=NULL){
                        p=p->next;
                        printf(" oid:%d",p->oid);
                        printf(" ti:%f",p->ti);
                        printf(" x1:%f",p->x1);
                        printf(" y1:%f",p->y1);
                        printf(" x2:%f",p->x2);
                        printf("y2:%f",p->y2);
                        printf("p:%i\n",p->p);

            }
            menu();
}

//estadistica de objetos
void estadistica(){
      int
estaticos=0,moviles=0,totales2=0,todos=0,TotalDinamicos=0,totalesd
=0;

      char   buffer[80];
      char  buffer2[80];
      char  buffer3[80];
      char  buffer4[80];
      char  buffer5[80];
      char  buffer6[80];
      char  buffer7[80];
      char buffer8[80];

      struct objeto *r,*p,*q,*headobjeto;
```

```
      int i;
      FILE *ptr;

      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);



      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;//llpp de todos los objetos del set

      //abrir archivo set para pasarlo a la llpp
      ptr = fopen ("set", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo set");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);

                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);
```

```
           p=headobjeto;
           r,q;
           printf("\n**********buscando objetos se mueven o/y
cambian de forma.*************");


                     p=p->next;
                     q=p->next;

           FILE *ptr4;//creamos archivo para guardar objetos mas
moviles
           ptr4 = fopen("moviles","w+");


           if(ptr4==NULL){
                printf ("\nError al crear Archivo moviles");
                menu();
                      }



           //buscando los objetos que sean mas moviles al menos 1
vez
           while(p->next!=NULL){
           todos++;

                     while(q!=NULL){

           //el mismo oid y que las coordenadas son  distintas, o
la X o l Y
                     if((p->oid==q->oid)&&((p->x1!=q->x1)||(p-
>y1!=q->y1)))
                     {
                     fprintf(ptr4,"%d %f %f %f %f %f %d\n",q-
>oid,q->ti,q->x1,q->y1,q->x2,q->y2,q->p);
                     moviles++;
                     }
                     q=q->next;
                     }
           p=p->next;
           q=p->next;
           }
           fclose(ptr4);



////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
/////////////////////////////
           struct objeto
           *r2,*p2,*q2,*headobjeto2; FILE *ptr2;
```

```
/* llenar los arreglos con '\0' para facilitar programacion
      */ memset (buffer, '\0', 80);
      memset (buffer2, '\0', 80);
      memset (buffer3, '\0', 80);
      memset (buffer4, '\0', 80);
      memset (buffer5, '\0', 80);
      memset (buffer6, '\0', 80);
      memset (buffer7, '\0', 80);
      memset (buffer8, '\0', 80);


      headobjeto2=newobjeto();
      p2=headobjeto2;
      headobjeto2->next=NULL;//llpp de todos los objetos moviles

      ptr2 = fopen ("moviles", "r");

      if(ptr2==NULL){
                printf ("\nNo existe el archivo moviles");
                menu();
      }
      while(feof(ptr2)==0){
                      q2=newobjeto();

                      fgets (buffer, 80,ptr2);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q2->oid=atoi(buffer2);
                      q2->ti=atof(buffer3);
                      q2->x1=atof(buffer4);
                      q2->y1=atof(buffer5);
                      q2->x2=atof(buffer6);
                      q2->y2=atof(buffer7);
                      q2->p=atoi(buffer8);

                                p2->next=q2;
                                r2=p2; p2=q2;
                                q2-
                                >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r2->next=NULL;

            fclose (ptr2);

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////////////
/////////////////////////////////////////////////buscando
inmoviles o
```

```
estaticos//////////////////////////////////////////////////////////
//////
           int cont=0,totales=0;
           //llpp de los objetos
           moviles p2=headobjeto2;
           r2,q2;
           p2=p2->next;
           q2=p2;
           //llpp de los objetos del
           set p=headobjeto;
           r,q; p=p-
           >next;
           q=p->next;
           r=p2;
           //lista total de todos los objetos
           moviles FILE *ptre;
           ptre = fopen ("estaticos", "w+");

           if(ptre==NULL){
                printf ("\nError al crear archivo estaticos");
                menu();
           }
           while(r!=NULL)
                {
                totales++;

                r=r->next;

                }
           while(p!=NULL){

                      cont=0;
                      while(p2!=NULL){

                      //si se encuentra en la lista de objetos
moviles sera impreso en totalmoviles
                      if(p->oid!=p2->oid)
                      {
                      cont++;

                      }
                      p2=p2->next;
                      }

                      if(cont==totales){
                      fprintf(ptre,"%d %f %f %f %f %f %d\n",p-
>oid,p->ti,p->x1,p->y1,p->x2,p->y2,p->p);
                      estaticos++;
                      }

           p=p->next;
           p2=headobjeto2->next;
```

```
            }
            fclose(ptre);

////////////////////////////////////////////////total
Moviles/////////////////////////////////////////////

            struct objeto
            *r3,*p3,*q3,*headobjeto3; FILE *ptr3;


            //pedimos memoria para una 3ra
            llpp headobjeto3=newobjeto();
               p3=headobjeto3;
               headobjeto3->next=NULL;//llpp de todos los objetos
estaticos


                  //llpp de los objetos del
            set p=headobjeto;
            r,q; p=p-
            >next;
            q=p->next;
            printf("\np->oid:%d lalollaoalaoa",p->oid);
            //llpp de los objetos estaticos

            ptr3 = fopen ("estaticos", "r");

            if(ptr3==NULL){
                  printf ("\nNo existe el archivo estaticos");
                  menu();
            }
            while(feof(ptr3)==0){
                        q3=newobjeto();

                        fgets (buffer, 80,ptr3);
                        sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                        q3->oid=atoi(buffer2);
                        q3->ti=atof(buffer3);
                        q3->x1=atof(buffer4);
                        q3->y1=atof(buffer5);
                        q3->x2=atof(buffer6);
                        q3->y2=atof(buffer7);
                        q3->p=atoi(buffer8);

                                    p3->next=q3;
                                    r3=p3; p3=q3;
                                    q3-
                                    >next=NULL;

            }
            //el ultimo de la lista va repetido lo borramos
```

```
            r3->next=NULL;

            fclose(ptr3);
            p3=headobjeto3->next;

            while(p3!=NULL){
            p3=p3->next;
            totalesd++;
                  }



            //lista total de todos los objetos
            moviles FILE *ptrd;
            ptrd = fopen ("TotalMoviles", "w+");

            if(ptrd==NULL){
                  printf ("\nError al crear archivo TotalMoviles");
                  menu();
            }


            p3=headobjeto3->next;
            p=headobjeto->next;
            int cont2=0;
            while(p!=NULL){

                        cont2=0;
                        while(p3!=NULL){

                        //si se encuentra en la lista de objetos
moviles sera impreso en totalmoviles
                        if(p->oid!=p3->oid)
                        {
                        cont2++;

                        }
                        p3=p3->next;
                        }

                        if(cont2==totalesd){
                        fprintf(ptrd,"%d %f %f %f %f %f %d\n",p-
>oid,p->ti,p->x1,p->y1,p->x2,p->y2,p->p);
                        TotalDinamicos++;
                        }

            p=p->next;
            p3=headobjeto3->next;
            }
```

```
          fclose(ptrd);



////////////////////////////////////////////////////////////////////
//////////////////////////////////
printf("\nGenerando archivos con objetos mas moviles, el
totalmoviles y estaticos......\n\n");
////////////////////////////////////////////////generar
estadisticas
          todos++;
          totales2=moviles+estaticos;
          printf("\n todos los objetos existentes en el set :
%d",todos);
          printf("\n total : %d",totales2);
          printf("\n movimientos:%d",moviles);
          printf("\n Objetos Dinamicos:%d",TotalDinamicos);
          printf("\n Objetos estaticos:%d",estaticos);
          printf("\n generando archivos con objetos mas
moviles......\n\n");

     menu();
}

void distribucion(){
           int proc,me,procesadores;


          // MPI_Comm_size(MPI_COMM_WORLD, &proc);
          //MPI_Comm_rank(MPI_COMM_WORLD, &me);

          printf("\nIngrese numero de
          procesadores:"); scanf("%d",&procesadores);

           printf("version para %d
           procesos...\n",procesadores); switch(procesadores){
                                case 1:distribuir1();
                                     break;
                                case 2:distribuir2();
                                     break;
                                case 4:distribuir4();

                                     break;
                                case 8:distribuir8();
                                     break;
                                case 16:distribuir16();
                                     break;
                                //case 32:distribuir32();
                                // break;


           default:
```

```
                              printf("no es una opcion...version para
2,4,8,16 o 32 procesos\n");
                              printf("presione <ENTER>para
continuar:\n");
                              getchar();
                              getchar();
                              menu();
                              break;
                              }


}
void distribuir1(){

char buffer[80];
      char  buffer2[80];
      char  buffer3[80];
      char  buffer4[80];
      char  buffer5[80];
      char  buffer6[80];
      char  buffer7[80];
      char buffer8[80];

      struct objeto
      *r,*p,*q,*headobjeto; int i;


      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo con el total de los moviles
      FILE *ptr;
      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("TotalMoviles", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo moviles");
                menu();
      }
      while(feof(ptr)==0){
                     q=newobjeto();
```

```
                        fgets (buffer, 80,ptr);
                        sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                        q->oid=atoi(buffer2);
                        q->ti=atof(buffer3);
                        q->x1=atof(buffer4);
                        q->y1=atof(buffer5);
                        q->x2=atof(buffer6);
                        q->y2=atof(buffer7);
                        q->p=atoi(buffer8);


                                    p->next=q;
                                    r=p;
                                    p=q; q-
                                    >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;
      FILE *ptr1;


      ptr1 = fopen ("./data_set/mov_10/1/set1", "w+");




      if(ptr1==NULL){
                  printf ("\nNo existe el archivo set1");
                  menu();
      }


      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}


      }
```

```
      //cerrando archivo
      fclose (ptr1);
////////////////////////////////////////////////añadir los
estaticos/////////////////////////////////////////////////////////
////


      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo estaticos

      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("estaticos", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo estaticos");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;
```

```
            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;

      ptr1 = fopen ("./data_set/mov_10/2/set1", "a+");


      if(ptr1==NULL){
                  printf ("\nNo existe el archivo set1");
                  menu();
      }


      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      }
fclose (ptr1);
printf ("\nGenerada la distribucion circular en 2
      archivos"); menu();

}

void distribuir2(){

      char   buffer[80];
      char  buffer2[80];
      char  buffer3[80];
      char  buffer4[80];
      char  buffer5[80];
      char  buffer6[80];
      char  buffer7[80];
      char buffer8[80];

      struct objeto
      *r,*p,*q,*headobjeto; int i;


      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset (buffer, '\0', 80);
```

```
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo con el total de los moviles
      FILE *ptr;
      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("TotalMoviles", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo moviles");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;
      FILE *ptr1,*ptr2;
```

```
      ptr1 = fopen ("./data_set/mov_10/2/set1", "w+");
      ptr2 = fopen ("./data_set/mov_10/2/set2", "w+");



      if(ptr1==NULL){
                printf ("\nNo existe el archivo set1");
                menu();
      }
      if(ptr2==NULL){
                printf ("\nNo existe el archivo set2");
                menu();
      }


      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}


      }


      //cerrando archivo
      fclose (ptr1);
      fclose (ptr2);
//////////////////////////////////////////////////añadir los
estaticos//////////////////////////////////////////////////////////
////



      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);
```

```
      //abrir archivo estaticos

      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("estaticos", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo estaticos");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;

      ptr1 = fopen ("./data_set/mov_10/2/set1", "a+");
      ptr2 = fopen ("./data_set/mov_10/2/set2", "a+");


      if(ptr1==NULL){
                printf ("\nNo existe el archivo set1");
                menu();
      }
      if(ptr2==NULL){
                printf ("\nNo existe el archivo set2");
```

```
                menu();
      }


      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      }
fclose (ptr1);
fclose (ptr2);
printf ("\nGenerada la distribucion circular en 2
      archivos"); menu();

}

void distribuir4(){


      //int proc,me;
      //MPI_Comm_size(MPI_COMM_WORLD, &proc);
      //MPI_Comm_rank(MPI_COMM_WORLD, &me);




      char   buffer[80];
      char  buffer2[80];
      char  buffer3[80];
      char  buffer4[80];
      char  buffer5[80];
      char  buffer6[80];
      char  buffer7[80];
      char buffer8[80];

      struct objeto
      *r,*p,*q,*headobjeto; int i;


      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset (buffer, '\0', 80);
      memset (buffer2, '\0', 80);
```

```
      memset (buffer3, '\0', 80);
      memset (buffer4, '\0', 80);
      memset (buffer5, '\0', 80);
      memset (buffer6, '\0', 80);
      memset (buffer7, '\0', 80);
      memset (buffer8, '\0', 80);

      //abrir archivo
      FILE *ptr;
      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("TotalMoviles", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo moviles");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;

      FILE *ptr1,*ptr2,*ptr3,*ptr4;
```

```
      ptr1  =  fopen  ("./data_set/mov_10/4/set1",  "w+");
      ptr2  =  fopen  ("./data_set/mov_10/4/set2",  "w+");
      ptr3  =  fopen  ("./data_set/mov_10/4/set3",  "w+");
      ptr4 = fopen ("./data_set/mov_10/4/set4", "w+");

      if(ptr1==NULL){
                printf ("\nNo existe el archivo set1");
                menu();
      }
      if(ptr2==NULL){
                printf ("\nNo existe el archivo set2");
                menu();
      }
      if(ptr3==NULL){
                printf ("\nNo existe el archivo set3");
                menu();
      }
      if(ptr4==NULL){
                printf ("\nNo existe el archivo set4");
                menu();
      }

      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr3,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr4,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      }

fclose (ptr1);
fclose (ptr2);
fclose (ptr3);
fclose (ptr4);
```

```
//////////////////////////////////////////////////añadir los
estaticos//////////////////////////////////////////////////////////
////



      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo

      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("estaticos", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo estaticos");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;
```

```
            fclose (ptr);


      p=headobjeto;
      p=p->next;



      ptr1 = fopen ("./data_set/mov_10/4/set1", "a+");
      ptr2 = fopen ("./data_set/mov_10/4/set2", "a+");
      ptr3 = fopen ("./data_set/mov_10/4/set3", "a+");
      ptr4 = fopen ("./data_set/mov_10/4/set4", "a+");

      if(ptr1==NULL){
                  printf ("\nNo existe el archivo set1");
                  menu();
      }
      if(ptr2==NULL){
                  printf ("\nNo existe el archivo set2");
                  menu();
      }
      if(ptr3==NULL){
                  printf ("\nNo existe el archivo set3");
                  menu();
      }
      if(ptr4==NULL){
                  printf ("\nNo existe el archivo set4");
                  menu();
      }

      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr3,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
```

```
      fprintf(ptr4,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      }
      fclose  (ptr1);
      fclose  (ptr2);
      fclose  (ptr3);
      fclose (ptr4);
      printf ("\nGenerada la distribucion circular en 4
      archivos"); menu();

}

void distribuir8(){


      //int proc,me;
      //MPI_Comm_size(MPI_COMM_WORLD, &proc);
      //MPI_Comm_rank(MPI_COMM_WORLD, &me);




      char   buffer[80];
      char  buffer2[80];
      char  buffer3[80];
      char  buffer4[80];
      char  buffer5[80];
      char  buffer6[80];
      char  buffer7[80];
      char buffer8[80];

      struct objeto
      *r,*p,*q,*headobjeto; int i;


      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo
      FILE *ptr;
      headobjeto=newobjeto();
```

```
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("TotalMoviles", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo moviles");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                    p->next=q;
                                    r=p;
                                    p=q; q-
                                    >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;

      FILE *ptr1,*ptr2,*ptr3,*ptr4,*ptr5,*ptr6,*ptr7,*ptr8;


      ptr1  =  fopen  ("./data_set/mov_10/8/set1",  "w+");
      ptr2  =  fopen  ("./data_set/mov_10/8/set2",  "w+");
      ptr3  =  fopen  ("./data_set/mov_10/8/set3",  "w+");
      ptr4  =  fopen  ("./data_set/mov_10/8/set4",  "w+");
      ptr5  =  fopen  ("./data_set/mov_10/8/set5",  "w+");
      ptr6  =  fopen  ("./data_set/mov_10/8/set6",  "w+");
      ptr7  =  fopen  ("./data_set/mov_10/8/set7",  "w+");
      ptr8 = fopen ("./data_set/mov_10/8/set8", "w+");
```

```
      if(ptr1==NULL){
                printf ("\nNo existe el archivo set1");
                menu();
      }
      if(ptr2==NULL){
                printf ("\nNo existe el archivo set2");
                menu();
      }
      if(ptr3==NULL){
                printf ("\nNo existe el archivo set3");
                menu();
      }
      if(ptr4==NULL){
                printf ("\nNo existe el archivo set4");
                menu();
      }
      if(ptr5==NULL){
                printf ("\nNo existe el archivo set5");
                menu();
      }
      if(ptr6==NULL){
                printf ("\nNo existe el archivo set6");
                menu();
      }
      if(ptr7==NULL){
                printf ("\nNo existe el archivo set7");
                menu();
      }
      if(ptr8==NULL){
                printf ("\nNo existe el archivo set8");
                menu();
      }



      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr3,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
```

```
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr4,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr5,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr6,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr7,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr8,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      }

fclose (ptr1);
fclose (ptr2);
fclose (ptr3);
fclose (ptr4);
fclose (ptr5);
fclose (ptr6);
fclose (ptr7);
fclose (ptr8);
//////////////////////////////////////////////////añadir los
estaticos////////////////////////////////////////////////////////
////



      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset (buffer, '\0', 80);
      memset (buffer2, '\0', 80);
```

```
      memset (buffer3, '\0', 80);
      memset (buffer4, '\0', 80);
      memset (buffer5, '\0', 80);
      memset (buffer6, '\0', 80);
      memset (buffer7, '\0', 80);
      memset (buffer8, '\0', 80);

      //abrir archivo

      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("estaticos", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo estaticos");
                menu();
      }
      while(feof(ptr)==0){
                      q=newobjeto();

                      fgets (buffer, 80,ptr);
                      sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                      q->oid=atoi(buffer2);
                      q->ti=atof(buffer3);
                      q->x1=atof(buffer4);
                      q->y1=atof(buffer5);
                      q->x2=atof(buffer6);
                      q->y2=atof(buffer7);
                      q->p=atoi(buffer8);


                                  p->next=q;
                                  r=p;
                                  p=q; q-
                                  >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;
```

```
      ptr1  =  fopen  ("./data_set/mov_10/8/set1",  "a+");
      ptr2  =  fopen  ("./data_set/mov_10/8/set2",  "a+");
      ptr3  =  fopen  ("./data_set/mov_10/8/set3",  "a+");
      ptr4  =  fopen  ("./data_set/mov_10/8/set4",  "a+");
      ptr5  =  fopen  ("./data_set/mov_10/8/set5",  "a+");
      ptr6  =  fopen  ("./data_set/mov_10/8/set6",  "a+");
      ptr7  =  fopen  ("./data_set/mov_10/8/set7",  "a+");
      ptr8 = fopen ("./data_set/mov_10/8/set8", "a+");

      if(ptr1==NULL){
                printf ("\nNo existe el archivo set1");
                menu();
      }
      if(ptr2==NULL){
                printf ("\nNo existe el archivo set2");
                menu();
      }
      if(ptr3==NULL){
                printf ("\nNo existe el archivo set3");
                menu();
      }
      if(ptr4==NULL){
                printf ("\nNo existe el archivo set4");
                menu();
      }
      if(ptr5==NULL){
                printf ("\nNo existe el archivo set5");
                menu();
      }
      if(ptr6==NULL){
                printf ("\nNo existe el archivo set6");
                menu();
      }
      if(ptr7==NULL){
                printf ("\nNo existe el archivo set7");
                menu();
      }
      if(ptr8==NULL){
                printf ("\nNo existe el archivo set8");
                menu();
      }


      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
```

```
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr3,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr4,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr5,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr6,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr7,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr8,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      }

      fclose  (ptr1);
      fclose  (ptr2);
      fclose  (ptr3);
      fclose  (ptr4);
      fclose  (ptr5);
      fclose  (ptr6);
      fclose  (ptr7);
      fclose (ptr8);
      printf ("\nGenerada la distribucion circular en 8
      archivos"); menu();
```

```
}


void distribuir16(){


      //int proc,me;
      //MPI_Comm_size(MPI_COMM_WORLD, &proc);
      //MPI_Comm_rank(MPI_COMM_WORLD, &me);




      char   buffer[80];
      char  buffer2[80];
      char  buffer3[80];
      char  buffer4[80];
      char  buffer5[80];
      char  buffer6[80];
      char  buffer7[80];
      char buffer8[80];

      struct objeto
      *r,*p,*q,*headobjeto; int i;


      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo
      FILE *ptr;
      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("moviles", "r");

      if(ptr==NULL){
                printf ("\nNo existe el archivo moviles");
                menu();
      }
      while(feof(ptr)==0){
                     q=newobjeto();
```

```
                        fgets (buffer, 80,ptr);
                        sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                        q->oid=atoi(buffer2);
                        q->ti=atof(buffer3);
                        q->x1=atof(buffer4);
                        q->y1=atof(buffer5);
                        q->x2=atof(buffer6);
                        q->y2=atof(buffer7);
                        q->p=atoi(buffer8);


                                    p->next=q;
                                    r=p;
                                    p=q; q-
                                    >next=NULL;

            }
            //el ultimo de la lista va repetido lo
            borramos r->next=NULL;

            fclose (ptr);


      p=headobjeto;
      p=p->next;

      FILE
*ptr1,*ptr2,*ptr3,*ptr4,*ptr5,*ptr6,*ptr7,*ptr8,*ptr9,*ptr10,*ptr1
1,*ptr12,*ptr13,*ptr14,*ptr15,*ptr16;


      ptr1 = fopen ("./data_set/mov_10/16/set1", "w+");
      ptr2 = fopen ("./data_set/mov_10/16/set2", "w+");
      ptr3 = fopen ("./data_set/mov_10/16/set3", "w+");
      ptr4 = fopen ("./data_set/mov_10/16/set4", "w+");
      ptr5 = fopen ("./data_set/mov_10/16/set5", "w+");
      ptr6 = fopen ("./data_set/mov_10/16/set6", "w+");
      ptr7 = fopen ("./data_set/mov_10/16/set7", "w+");
      ptr8 = fopen ("./data_set/mov_10/16/set8", "w+");
      ptr9 = fopen ("./data_set/mov_10/16/set9", "w+");
      ptr10 = fopen ("./data_set/mov_10/16/set10", "w+");
      ptr11 = fopen ("./data_set/mov_10/16/set11", "w+");
      ptr12 = fopen ("./data_set/mov_10/16/set12", "w+");
      ptr13 = fopen ("./data_set/mov_10/16/set13", "w+");
      ptr14 = fopen ("./data_set/mov_10/16/set14", "w+");
      ptr15 = fopen ("./data_set/mov_10/16/set15", "w+");
      ptr16 = fopen ("./data_set/mov_10/16/set16", "w+");


      if(ptr1==NULL){
                  printf ("\nNo existe el archivo set1");
```

```
            menu();
}
if(ptr2==NULL){
            printf ("\nNo existe el archivo set2");
            menu();
}
if(ptr3==NULL){
            printf ("\nNo existe el archivo set3");
            menu();
}
if(ptr4==NULL){
            printf ("\nNo existe el archivo set4");
            menu();
}

if(ptr5==NULL){
            printf ("\nNo existe el archivo set5");
            menu();
}
if(ptr6==NULL){
            printf ("\nNo existe el archivo set6");
            menu();
}
if(ptr7==NULL){
            printf ("\nNo existe el archivo set7");
            menu();
}
if(ptr8==NULL){
            printf ("\nNo existe el archivo set8");
            menu();
}
if(ptr9==NULL){
            printf ("\nNo existe el archivo set9");
            menu();
}
if(ptr10==NULL){
            printf ("\nNo existe el archivo
            set10"); menu();
}
if(ptr11==NULL){
            printf ("\nNo existe el archivo
            set11"); menu();
}
if(ptr12==NULL){
            printf ("\nNo existe el archivo
            set12"); menu();
}

if(ptr13==NULL){
            printf ("\nNo existe el archivo
            set13"); menu();
}
```

```
      if(ptr14==NULL){
                printf ("\nNo existe el archivo
                set14"); menu();
      }
      if(ptr15==NULL){
                printf ("\nNo existe el archivo
                set15"); menu();
      }
      if(ptr16==NULL){
                printf ("\nNo existe el archivo
                set16"); menu();
      }

      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr3,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr4,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr5,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr6,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }
```

```
      if(p!=NULL){
      fprintf(ptr7,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr8,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr9,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr10,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr11,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr12,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr13,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr14,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr15,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
```

```
      }

      if(p!=NULL){
      fprintf(ptr16,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      }

fclose   (ptr1);
fclose   (ptr2);
fclose   (ptr3);
fclose   (ptr4);
fclose   (ptr5);
fclose   (ptr6);
fclose   (ptr7);
fclose   (ptr8);
fclose   (ptr9);
fclose  (ptr10);
fclose  (ptr11);
fclose  (ptr12);
fclose  (ptr13);
fclose  (ptr14);
fclose  (ptr15);
fclose (ptr16);
//////////////////////////////////////////////////añadir los
estaticos////////////////////////////////////////////////////////
////



      /* llenar los arreglos con '\0' para facilitar programacion
*/
      memset  (buffer,  '\0',  80);
      memset  (buffer2,  '\0',  80);
      memset  (buffer3,  '\0',  80);
      memset  (buffer4,  '\0',  80);
      memset  (buffer5,  '\0',  80);
      memset  (buffer6,  '\0',  80);
      memset  (buffer7,  '\0',  80);
      memset (buffer8, '\0', 80);

      //abrir archivo

      headobjeto=newobjeto();
      p=headobjeto;
      headobjeto->next=NULL;

      ptr = fopen ("estaticos", "r");

      if(ptr==NULL){
```

```
                printf ("\nNo existe el archivo estaticos");
                menu();
        }
        while(feof(ptr)==0){
                        q=newobjeto();

                        fgets (buffer, 80,ptr);
                        sscanf (buffer,"%s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8);
                        q->oid=atoi(buffer2);
                        q->ti=atof(buffer3);
                        q->x1=atof(buffer4);
                        q->y1=atof(buffer5);
                        q->x2=atof(buffer6);
                        q->y2=atof(buffer7);
                        q->p=atoi(buffer8);


                                        p->next=q;
                                        r=p;
                                        p=q; q-
                                        >next=NULL;

              }
              //el ultimo de la lista va repetido lo
              borramos r->next=NULL;

              fclose (ptr);


        p=headobjeto;
        p=p->next;




        ptr1 = fopen ("./data_set/mov_10/16/set1", "a+");
        ptr2 = fopen ("./data_set/mov_10/16/set2", "a+");
        ptr3 = fopen ("./data_set/mov_10/16/set3", "a+");
        ptr4 = fopen ("./data_set/mov_10/16/set4", "a+");
        ptr5 = fopen ("./data_set/mov_10/16/set5", "a+");
        ptr6 = fopen ("./data_set/mov_10/16/set6", "a+");
        ptr7 = fopen ("./data_set/mov_10/16/set7", "a+");
        ptr8 = fopen ("./data_set/mov_10/16/set8", "a+");
        ptr9 = fopen ("./data_set/mov_10/16/set9", "a+");
        ptr10 = fopen ("./data_set/mov_10/16/set10", "a+");
        ptr11 = fopen ("./data_set/mov_10/16/set11", "a+");
        ptr12 = fopen ("./data_set/mov_10/16/set12", "a+");
        ptr13 = fopen ("./data_set/mov_10/16/set13", "a+");
        ptr14 = fopen ("./data_set/mov_10/16/set14", "a+");
        ptr15 = fopen ("./data_set/mov_10/16/set15", "a+");
        ptr16 = fopen ("./data_set/mov_10/16/set16", "a+");
```

```
if(ptr1==NULL){
          printf ("\nNo existe el archivo set1");
          menu();
}
if(ptr2==NULL){
          printf ("\nNo existe el archivo set2");
          menu();
}
if(ptr3==NULL){
          printf ("\nNo existe el archivo set3");
          menu();
}
if(ptr4==NULL){
          printf ("\nNo existe el archivo set4");
          menu();
}

if(ptr5==NULL){
          printf ("\nNo existe el archivo set5");
          menu();
}
if(ptr6==NULL){
          printf ("\nNo existe el archivo set6");
          menu();
}
if(ptr7==NULL){
          printf ("\nNo existe el archivo set7");
          menu();
}
if(ptr8==NULL){
          printf ("\nNo existe el archivo set8");
          menu();
}

if(ptr9==NULL){
          printf ("\nNo existe el archivo set9");
          menu();
}
if(ptr10==NULL){
          printf ("\nNo existe el archivo
          set10"); menu();
}
if(ptr11==NULL){
          printf ("\nNo existe el archivo
          set11"); menu();
}
if(ptr12==NULL){
          printf ("\nNo existe el archivo
          set12"); menu();
}
```

```
      if(ptr13==NULL){
                printf ("\nNo existe el archivo
                set13"); menu();
      }
      if(ptr14==NULL){
                printf ("\nNo existe el archivo
                set14"); menu();
      }
      if(ptr15==NULL){
                printf ("\nNo existe el archivo
                set15"); menu();
      }
      if(ptr16==NULL){
                printf ("\nNo existe el archivo
                set16"); menu();
      }



      while(p!=NULL)
      {

      if(p!=NULL){
      fprintf(ptr1,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr2,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr3,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr4,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr5,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr6,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
```

```
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr7,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr8,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr9,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr10,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr11,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr12,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
      }

      if(p!=NULL){
      fprintf(ptr13,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr14,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr15,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;}

      if(p!=NULL){
      fprintf(ptr16,"%d %f %f %f %f %f %d\n",p->oid,p->ti,p->x1,p-
>y1,p->x2,p->y2,p->p);
      p=p->next;
```

```
      }

      }

      fclose   (ptr1);
      fclose   (ptr2);
      fclose   (ptr3);
      fclose   (ptr4);
      fclose   (ptr5);
      fclose   (ptr6);
      fclose   (ptr7);
      fclose   (ptr8);
      fclose   (ptr9);
      fclose  (ptr10);
      fclose  (ptr11);
      fclose  (ptr12);
      fclose  (ptr13);
      fclose  (ptr14);
      fclose  (ptr15);
      fclose (ptr16);
      printf ("\nGenerada la distribucion circular en 16
archivos");
      menu();

}


//MENU
void menu(){
            int opcion;

            printf("\n\n---ingrese la opcion deseada--- \n");
            printf("0---Crear Set\n");
            printf("1---mostrar objetos\n");
            printf("2---mostrar estadisticas\n");
            printf("3-distribuir objetos en archivos\n");
            printf("4---salir:\n");

            scanf(" %d",&opcion);
            switch(opcion){

                                    case 0: crearSet();
                                         break;
                                    case 1: leerobjeto();
                                         break;
                                    case 2:    estadistica();

                                         break;
                                    case 3 :distribucion();
                                         break;
                                    case 4 :
```

```
                                        exit(0);


            default:
                            printf("no es una opcion del menu\n");
                            printf("presione <ENTER>para
continuar:\n");
                            getchar();
                            getchar();
                            menu();
                            break;
                            }


}



//=================================================================
===========
//=  Function to generate Zipf (power law) distributed random
variables      =
//=    - Input: alpha and N
=
//=    - Output: Returns with Zipf distributed random variable
=
//=================================================================
===========
int zipf(double alpha, int n)
{
  static int first = TRUE;      // Static first time flag
  static double c = 0;          // Normalization constant
  double z;                     // Uniform random number (0 < z <
1)
  double sum_prob;              // Sum of probabilities
  double zipf_value;            // Computed exponential value to
be returned
  int    i;                     // Loop counter

  // Compute normalization constant on first call only
  if (first == TRUE)
  {
    for (i=1; i<=n; i++)
      c  =  c  +  (1.0  /  pow((double)  i,
    alpha)); c = 1.0 / c;
    first = FALSE;
  }

  // Pull a uniform random number (0 < z < 1)
  do
  {
```

```
    z = rand_val(0);
  }
  while ((z == 0) || (z == 1));

  // Map z to the value
  sum_prob = 0;
  for (i=1; i<=n; i++)
  {
    sum_prob  =  sum_prob  +  c  /  pow((double)  i,
    alpha); if (sum_prob >= z)
    {
      zipf_value    =
      i; break;
    }
  }

  // Assert that zipf_value is between 1 and N
  assert((zipf_value >=1) && (zipf_value <= n));

  return(zipf_value);
}

//=================================================================
=========
//= Multiplicative LCG for generating uniform(0.0, 1.0) random
numbers     =
//=   - x_n = 7^5*x_(n-1)mod(2^31 - 1)
=
//=   - With x seeded to 1 the 10000th x value should be
1043618065        =
//=   - From R. Jain, "The Art of Computer Systems Performance
Analysis," =
//=     John Wiley & Sons, 1991. (Page 443, Figure 26.2)
=
//=================================================================
=========
double rand_val(int seed)
{
  const long  a =      16807;  // Multiplier
  const long  m = 2147483647;  // Modulus
  const long  q =     127773;  // m div a
  const long  r =       2836;  // m mod a
  static long x;               // Random int value
  long        x_div_q;         // x divided by q
  long        x_mod_q;         // x modulo q
  long        x_new;           // New x value

  // Set the seed if argument is non-zero and then return
  zero if (seed > 0)
  {
    x = seed;
    return(0.0);
```

```
  }

  // RNG using integer arithmetic
  x_div_q = x / q;
  x_mod_q = x % q;
  x_new = (a * x_mod_q) - (r * x_div_q);
  if (x_new > 0)
    x = x_new;
  else
    x = x_new + m;

  // Return a random value between 0.0 and 1.0
  return((double) x / m);
}
```

genexp.c

```
//----- Include files ---------------------------------------------------------
#include <stdio.h>          // Needed for printf()
#include <stdlib.h>         // Needed for exit() and ato*()
#include <math.h>           // Needed for log()
#include <assert.h>         // Needed for assert() macro

#define NDEBUG


//----- Function prototypes --------------------------------------------------

double rand_val(int seed);   // Jain's RNG


//===== Main program ========================================================
int main(void)
{
  char   in_string[256];     // Input string
  FILE   *fp;                // File pointer to output file
  double lambda,x,z;         // Mean rate
  int exp_value;             // Exponential random variable
  int    num_values;         // Number of values
  int    i,n;                // Loop counter
```

```
// Output banner
printf("---------------------------------------- genexp.c ----- \n");
printf("- Program to generate exponential random variables - \n");
printf("------------------------------------------------------ \n");

// Prompt for output filename and then create/open the file

fp = fopen("numeros1.txt",
"w"); if (fp == NULL)
{
  printf("ERROR in creating output file (%s) \n",
  in_string); exit(1);
}

// Prompt for random number seed and then use it
//printf("Random number seed (greater than 0) ================>
"); //scanf("%s", in_string);
rand_val(1);

// Prompt for rate (lambda)
printf("Rate parameter (lambda) ============================>
"); scanf("%s", in_string);
lambda = atof(in_string);


// Prompt for number of values to generate
//printf("Number of values to generate =======================>
"); //scanf("%s", in_string);
num_values = 1000;


// Output message and generate interarrival times
printf("------------------------------------------------------ \n");
printf("- Generating samples to file                    - \n");
printf("------------------------------------------------------ \n");

// Generate and output exponential random
variables for (i=0; i<num_values; i++)
{

    do
```

```
    {
     z = rand_val(0);

    }
    while ((z == 0) || (z == 1));


     x=1.0 / lambda;

    // Compute exponential random variable using inversion method
     exp_value = (int)(-x * log(z));
     if(exp_value == 0 || exp_value>100)
     {
       i--;
     }
     else
     fprintf(fp, "%d\n", exp_value);
  }


  // Output message and close the output file
  printf("-------------------------------------------------------- \n");
  printf("- Done! \n");
  printf("-------------------------------------------------------- \n");
  fclose(fp);


}



//=========================================================================
//= Function to generate exponentially distributed random variables       =
//=    - Input: Mean value of distribution                                =
//=    - Output: Returns with exponentially distributed random variable   =
//=========================================================================

double rand_val(int seed)
{
  const long a = 16807; // Multiplier const
  long m = 2147483647; // Modulus const
  long q = 127773; // m div a
```

```
  const long r =       2836; // m mod a
  static long x;              // Random int value
  long       x_div_q;         // x divided by q
  long       x_mod_q;         // x modulo q
  long       x_new;           // New x value

  // Set the seed if argument is non-zero and then return
  zero if (seed > 0)
  {
    x = seed;
    return(0.0);
  }

  // RNG using integer
  arithmetic x_div_q = x / q;
  x_mod_q = x % q;
  x_new = (a * x_mod_q) - (r *
  x_div_q); if (x_new > 0)
    x = x_new;
  else
    x = x_new + m;
    printf("%d\n",x);
  // Return a random value between 0.0 and
  1.0 return((double) x / m);
}
```

generl.c

```
//----- Include files -------------------------------------------------------
#include <stdio.h>          // Needed for printf()
#include <stdlib.h>         // Needed for exit() and ato*()
#include <math.h>           // Needed for log()

//----- Constants -----------------------------------------------------------
#define MAX_STAGES 100      // Maximum number of stages

//----- Function prototypes -------------------------------------------------

double rand_val(int seed);     // Jain's RNG

//===== Main program ==========================================================
int main(void)
{
```

```
FILE    *fp;              // File pointer to output file
char    file_name[256];   // Output file name string
char    temp_string[256]; // Temporary string variable
double  lambda[MAX_STAGES]; // Mean of rates for stages
int exp_rv;               // Exponential random variable
int erl_rv;               // Erlang random variable
int     num_values;       // Number of values
int     num_stages;       // Number of stages
int     i,j,n;
 double z,beta;           // Loop counters

// Output banner
printf("----------------------------------------  generl.c ----- \n");
printf("- Program to generate Gamma random variables        - \n");
printf("-------------------------------------------------------  \n");

// Prompt for output filename and then create/open the
file fp = fopen("numeros2.txt", "w");
if (fp == NULL)
{
  printf("ERROR in creating output file (%s) \n",
  file_name); exit(1);
}

// Prompt for random number seed and then use it
//printf("Random number seed (greater than 0) ================> "); //For srand
function //scanf("%s", temp_string);
rand_val(1);

// Prompt for number of stages
printf("Number of stages\n [Gamma number is a sumatory of exponential numbers, number
of stages is the max number of the sumatory] ==================================> ");
scanf("%s", temp_string);
num_stages = atoi(temp_string);
if (num_stages > MAX_STAGES)
{
  printf("ERROR - too many stages (max stages is %d) \n",
  MAX_STAGES); exit(1);
}

// Prompt for stage rate and assign the rates
(lambda[]) printf("lambda =============> ");
```

```
scanf("%s", temp_string);
lambda[0] = atof(temp_string);
for (i=1; i<num_stages; i++)
lambda[i] = lambda[0];

// Prompt for number of values to generate
//printf("Number of values to generate ======================>
"); //scanf("%s", temp_string);
num_values = 1000;

//Output message and generate interarrival times
printf("------------------------------------------------------- \n");
printf("- Generating samples to file                    - \n");
printf("------------------------------------------------------- \n");

// Generate and output Erlang random variables
//  - Erlang random variable is a sum of exponential random variables
for (i=0; i<num_values; i++)//For to fprintf the gamma value
{
 erl_rv = 0;

 for (j=0; j<num_stages; j++) //Sumatory between 0 - num_stages of exponential values
 {
      beta = (1.0 / lambda[j]);

        do
      {
       z = rand_val(0);
      }
      while ((z == 0) || (z == 1));


  exp_rv = (int)((-1*beta) * log(z));
  erl_rv = erl_rv + exp_rv;
 }
 if(erl_rv == 0 || erl_rv > 100) //min value = 1 and max value = 100
 {
  i--;
 }
 else fprintf(fp, "%d \n", erl_rv);
}
```

```
  //Output message and close the output file
  printf("------------------------------------------------------- \n");
  printf("- Done! \n");
  printf("------------------------------------------------------- \n");
  fclose(fp);
}



double rand_val(int seed)
{
  const long a = 16807; // Multiplier const
  long m = 2147483647; // Modulus
  const long q =       127773; // m div a
  const long r =        2836; // m mod a
  static long x;              // Random int value
  long       x_div_q;         // x divided by q
  long       x_mod_q;         // x modulo q
  long       x_new;           // New x value

  // Set the seed if argument is non-zero and then return
  zero if (seed > 0)
  {
    x = seed;
    return(0.0);
  }

  // RNG using integer
  arithmetic x_div_q = x / q;
  x_mod_q = x % q;
  x_new = (a * x_mod_q) - (r *
  x_div_q); if (x_new > 0)
    x = x_new;
  else
    x = x_new + m;

  // Return a random value between 0.0 and
  1.0 return((double) x / m);
}
```

genzipf.c

```
//----- Include files -------------------------------------------------
#include <assert.h>          // Needed for assert() macro
#include <stdio.h>           // Needed for printf()
#include <stdlib.h>          // Needed for exit() and ato*()
#include <math.h>            // Needed for pow()

//----- Constants -----------------------------------------------------
#define FALSE          0    // Boolean false
#define TRUE           1    // Boolean true

//----- Function prototypes -------------------------------------------
int     zipf(double alpha, int n); // Returns a Zipf random variable
double  rand_val(int seed);        // Jain's RNG


//===== Main program ==========================================================
int main(void)
{
  FILE   *fp;               // File pointer to output file
  char   file_name[256];      // Output file name string
  char   temp_string[256];    // Temporary string variable
  double alpha;             // Alpha parameter
  double n;                 // N parameter
  int    num_values;        // Number of values
  int    zipf_rv;           // Zipf random variable
  int    i;                 // Loop counter
  struct distribucion *r,*p,*q,*headobjeto,*x;

  // Output banner
  printf("-------------------------------------------------------------------
                                        genzipf.c ----- \n");
  printf("-    Program to generate Zipf random variables       - \n");
  printf("-------------------------------------------------------------------
                                                          \n");

  // Prompt for output filename and then create/open the
  file fp = fopen("numeros3.txt", "w");
  if (fp == NULL)
  {
    printf("ERROR in creating output file (%s) \n",
    file_name); exit(1);
  }
```

```
// Prompt for random number seed and then use it
//printf("Random number seed (greater than 0) ================>
"); //scanf("%s", temp_string);
rand_val(1);

// Prompt for alpha value
printf("Alpha value ========================================>
"); scanf("%s", temp_string);
alpha = atof(temp_string);

// Prompt for N value
//printf("N value ============================================>
"); //scanf("%s", temp_string);
n = 100;

// Prompt for number of values to generate
//printf("Number of values to generate =======================>
"); //scanf("%s", temp_string);
num_values = 1000;

// Output "generating" message
printf("------------------------------------------------------ \n");
printf("- Generating samples to file                          - \n");
printf("------------------------------------------------------ \n");


for (i=0; i<num_values; i++)
{
  zipf_rv = zipf(alpha, n);

  fprintf(fp, "%d \n", zipf_rv);
}

// Output "done" message and close the output file
printf("------------------------------------------------------ \n");
printf("- Done! \n");
printf("------------------------------------------------------ \n");


fclose(fp);
```

```
}


int zipf(double alpha, int n)
{
 static int first = TRUE;     // Static first time flag
 static double c = 0;          // Normalization constant
 double z;                     // Uniform random number (0 < z < 1)
 double sum_prob;              // Sum of probabilities
 double zipf_value;            // Computed exponential value to be returned
 int  i;                       // Loop counter

 // Compute  normalization  constant  on  first  call
 only if (first == TRUE)
 {
  for (i=1; i<=n; i++)
   c  =  c  +  (1.0  /  pow((double)  i,
 alpha)); c = 1.0 / c;
 first = FALSE;
 }

 // Pull a uniform random number (0 < z < 1)
 do
 {
  z = rand_val(0);
 }
 while ((z == 0) || (z == 1));

 // Map  z  to  the
 value sum_prob = 0;
 for (i=1; i<=n; i++)
 {
  sum_prob  =  sum_prob  +  c  /  pow((double)  i,
  alpha); if (sum_prob >= z)
  {
   zipf_value = i;
   break;
  }
 }

 // Assert  that  zipf_value  is  between  1  and  N
 assert((zipf_value >=1) && (zipf_value <= n));
```

```
  return(zipf_value);
}

//========================================================================
//= Multiplicative LCG for generating uniform(0.0, 1.0) random numbers    =
//=   - x_n = 7^5*x_(n-1)mod(2^31 - 1)                                    =
//=   - With x seeded to 1 the 10000th x value should be 1043618065       =
//=   - From R. Jain, "The Art of Computer Systems Performance Analysis," =
//=     John Wiley & Sons, 1991. (Page 443, Figure 26.2)                  =
//========================================================================
double rand_val(int seed)
{
  const long a = 16807; // Multiplier const
  long m = 2147483647; // Modulus
  const long q =       127773; // m div a
  const long r =        2836; // m mod a
  static long x;              // Random int value
  long       x_div_q;         // x divided by q
  long       x_mod_q;         // x modulo q
  long       x_new;           // New x value

  // Set the seed if argument is non-zero and then return
  zero if (seed > 0)
  {
    x = seed;
    return(0.0);
  }

  // RNG using integer
  arithmetic x_div_q = x / q;
  x_mod_q = x % q;
  x_new = (a * x_mod_q) - (r *
  x_div_q); if (x_new > 0)
    x = x_new;
  else
    x = x_new + m;

  // Return a random value between 0.0 and
  1.0 return((double) x / m);
}
```

GenQuery.c

```
#include <assert.h>          // Needed for assert() macro
#include <stdio.h>           // Needed for printf()
#include <stdlib.h>          // Needed for exit() and ato*()
#include <math.h>            // Needed for pow()

//estructura.
struct distribucion
{
  int numero;
  float x1;
  float y1;
  float x2;
  float y2;
  int ti;
  int tf;
  int veces;
  struct objeto *next;
};

//reserva de memoria recursiva struct
distribucion *distribucion(); struct
distribucion *newdistribucion(){
return((struct distribucion *)malloc(sizeof(struct distribucion)));
}


//===== Main program ========================================================
int main(void)
{
  FILE   *fp;               // File pointer to output file
  FILE *ptr;
  FILE *ptr2;
  char   file_name[256];       // Output file name string
  char   temp_string[256];     // Temporary string variable
  double alpha;              // Alpha parameter
  double n;

  int    num_values;          // Number of values
  int    zipf_rv;            // Zipf random variable
  int    i,query_num,cont_ti,cont_ts,as;                // Loop counter
```

```
struct distribucion
*r,*p,*q,*headobjeto,*x; int times;
char    buffer[80];
char  buffer2[80];
char  buffer3[80];
char  buffer4[80];
char  buffer5[80];
char  buffer6[80];
char buffer7[80];


char number[80];
char number2[80];
char ruta1[50];
char ruta2[50];


// // Prompt for output filename and then create/open the file
// printf("Output file name ==================================> ");
// scanf("%s", file_name);




//Start querys files.
int consultas=1;

while(consultas<=3){ //While for the 3
files. int numeros=1;

while(numeros<=3){ //generador de consultas para cada experimento

      sprintf(ruta1,"numeros%d.txt",numeros); //lee los numeros generados por las distintas
distribuciones
       ptr2 = fopen (ruta1, "r");

       if(ptr2==NULL){
          printf ("\nFile Doesn't Exist");
       }

       printf("------------------------------------------------------ \n");
```

```
printf("- Generating query                    - \n");
printf("------------------------------------------------------- \n");
memset (buffer, '\0', 80);
memset (buffer2, '\0', 80);
memset (buffer3, '\0', 80);
memset (buffer4, '\0', 80);
memset (buffer5, '\0', 80);
memset (buffer6, '\0', 80);
memset (buffer7, '\0', 80);


memset (number, '\0', 80);
memset (number2, '\0', 80);

headobjeto=newdistribucion();
p=headobjeto;
headobjeto->next=NULL;

i=0;
sprintf(file_name,"./querys/queryresult%d.%d.txt",consultas,numeros);
fp = fopen(file_name, "w");
while(feof(ptr2)==0){//Read line per line the file "numeros.txt"



  fgets (number, 80,ptr2);
  sscanf (number,"%s",number2);

  printf("numero del archivo numeros: %s \n",number2 );

 q=newdistribucion();

 if (fp == NULL)
 {
  printf("ERROR in creating output file (%s) \n", file_name);
  exit(1);
 }
  if(i==0)
  {
        query_num=1;
        sprintf(ruta2,"./querys/query%d.txt",consultas);
        ptr = fopen (ruta2, "r");
```

```
            if(ptr==NULL){
               printf ("\nNo existe el archivo de consultas");
            }
          while(feof(ptr)==0){//imprime en los archivos las consultas
            seleccionadas fgets (buffer, 80,ptr);
            sscanf (buffer,"%s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7);
            // printf("tipo del archivo consultas: %s \n",buffer7 );
           if (query_num == atoi(number2)) {
            // printf("agregado\n");
             q->numero = atoi(number2);
             q->x1 = atof(buffer2);
             q->y1 = atof(buffer3);
             q->x2 = atof(buffer4);
             q->y2 = atof(buffer5);
             q->ti = atoi(buffer6);
             q->tf = atoi(buffer7);
             q->veces = 1; p-
             >next=q;
              r=p;
              p=q; q-
              >next=NULL;
              fprintf(fp, "%f %f %f %f %d %d\n", q->x1,q->y1,q->x2,q->y2,q->ti,q->tf);


           }

            query_num++;


          }
            fclose(ptr);

       }

       if(i!=0){

        x=headobjeto->next;
        int contador = 0;

        while(x!=NULL)
        {
```

```
 if(x->numero==atoi(number2))
 {
  x->veces=x->veces+1;
  contador+=1;
 }
 x=x->next;
}

if(contador==0)
{
        sprintf(ruta2,"./querys/query%d.txt",consultas);
        ptr = fopen (ruta2, "r");
        if(ptr==NULL){
           printf ("\nNo existe el archivo de consultas");
        }

      query_num=1;
      while(feof(ptr)==0){//imprime en el archivo las consultas
        seleccionadas fgets (buffer, 80,ptr);
        sscanf (buffer,"%s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7);
        // printf("tipo del archivo consultasss: %s \n",buffer7 );
       if (query_num == atoi(number2)){
          q->numero = atoi(number2);
          q->x1 = atof(buffer2); q->y1
          = atof(buffer3); q->x2 =
          atof(buffer4); q->y2 =
          atof(buffer5);
          q->ti = atoi(buffer6);
          q->tf = atoi(buffer7);
          q->veces = 1; p-
          >next=q;
          r=p;
          p=q; q-
          >next=NULL;
          fprintf(fp, "%f %f %f %f %d %d\n", q->x1,q->y1,q->x2,q->y2,q->ti,q->tf);

       }
       query_num++;

      }
```

```
                fclose(ptr);

            }
            else{
               sprintf(ruta2,"./querys/query%d.txt",consultas);
                    ptr = fopen (ruta2, "r");
                    if(ptr==NULL){
                       printf ("\nNo existe el archivo de consultas");
                    }

                  query_num=1;
                  while(feof(ptr)==0){
                    fgets (buffer, 80,ptr);
                     sscanf (buffer,"%s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7);
                     // printf("tipo del archivo consultasss: %s \n",buffer7 );
                    if (query_num == atoi(number2)){

                        fprintf(fp, "%f %f %f %f %d %d\n",
atof(buffer2),atof(buffer3),atof(buffer4),atof(buffer5),atoi(buffer6),atoi(buffer7));

                    }
                    query_num++;

                  }

                    fclose(ptr);

             }

           }
            if(i==700) //crea archivo de las ultimas 300 consultas
            {
              fclose(fp);
              sprintf(file_name,"./querys/2queryresult%d.%d.txt",consultas,numeros);
              fp = fopen(file_name, "w");
                  }
           i++;
         }

       cont_ti=0;
```

```
        cont_ts=0;
          p=headobjeto->next;
          printf("\n**********LISTA DE CONSULTAS**********\n");

           while(p!=NULL){//imprime las consultas

              printf(" numero:%d ",p->numero);
              printf(" x1:%f ",p->x1);
              printf(" y1:%f ",p->y1);
              printf(" x2:%f ",p->x2);
              printf(" y2:%f ",p->y2);
              printf(" ti:%d ",p->ti); printf("
              tf:%d ",p->tf); printf("
              veces:%d ",p->veces); if (p-
              >ti!=p->tf){
              cont_ti=cont_ti+p->veces;
              printf(" tipo: ti\n");

              }
              else { cont_ts=cont_ts+p-
                >veces; printf(" tipo:
                ts\n");
              }
              p=p->next;

           }
           fclose(fp);
          // Output "done" message and close the output file
          printf("------------------------------------------------------ \n");
          printf("- Total de consulta TI= %d \n",cont_ti);
          printf("- Total de consulta TS= %d \n",cont_ts);
          printf("------------------------------------------------------ \n");
           fclose(ptr2);
           numeros++;
    }
     consultas++;

}

}
```

Modificación archivo st.cpp del algoritmo de yu fei tao.

```
/*******************************
MVRTimeQuery: This funciton to conduct a series of queries on the give MVR-tree
Para:
  mvr_tree: The mvrtree to be queried
  on query_fname: The query file name
  count: The number of results retrieved

  return value: The number of I/O's
Precondition:
  The mvr-tree should have been loaded and the query file exists
Postcondition:
  None.
*******************************/

int MVRTimeQuery(MVRTree *mvr_tree, char *query_fname, int &count, int maq)
{
    FILE *arch,*arch2,*arch3,*arch4,*arch5;
    char out[50],out2[50],letra;
    int buffer,setqueryres,ios,flagquery,records;
   float v2;
    char  buffer1[80];
    char  buffer2[80];
    char  buffer3[80];
    char  buffer4[80];
    char  buffer5[80];
    char  buffer6[80];
    char  buffer7[80];
    char  buffer8[80];
    char buffer9[80];

    FILE *query_file;
    query_file = fopen(query_fname, "r");

   if (query_file == NULL)
            error("The query file does not exist\n", true);

    sprintf(out, "./salidas/flag%d", maq+1);

     arch3 = fopen(out, "r");
```

```
if (arch3 == NULL){
        printf("no esta el archivo bandera\n");
        buffer=0;


}
else{
        fscanf(arch3, "%d",
        &buffer); fclose(arch3);

}
```

```
float *f_data = new float
[4]; int *i_data = new int
[4]; int f_num, i_num;

int   retrieve_count   =
0;  int  query_count  =
0; int contTs = 0;
int contTi = 0;

int old_io_count = mvr_tree -> file -> get_iocount(); //obtención de nodos accesados antes
de realizar la consulta
sprintf(out, "./salidas/proc%d.out",
maq+1); arch = fopen(out,"w");
sprintf(out2, "./salidas/resquery%d",
maq+1); arch4 = fopen(out2, "r");

        if (arch4 == NULL){
        setqueryres=0;
        }
        else{
        setqueryres=1;
        fclose(arch4);
        }
  memset (buffer1, '\0', 80);
  memset (buffer2, '\0', 80);
```

```
        memset (buffer3, '\0', 80);
        memset  (buffer4,'\0',  80);
        memset (buffer5, '\0', 80);
        memset (buffer6, '\0', 80);
        memset (buffer7, '\0', 80);
        memset (buffer8, '\0', 80);
        memset (buffer9, '\0', 80);


    while (read_QueryGen_record(query_file, f_data, f_num, i_data, i_num))//lectura de
archivo de consultas
    {

                SortedMVRLinList *result_list = new SortedMVRLinList();


                char *retrieved_record = new char[OBJECTNUM];
                for (int i = 0; i < OBJECTNUM; i++)
                        retrieved_record [i] = 0;


                float *query_mbr = new float [7]; //guarda las consultas en los vectores
                query_mbr [0] = f_data [0];
                query_mbr  [1]  =  f_data  [1];
                query_mbr  [2]  =  f_data  [2];
                query_mbr  [3]  =  f_data  [3];
                query_mbr  [4]  =  i_data  [0];
                query_mbr [5] = i_data [1];



                flagquery=0;
                ios=0;//variable que calcula el ahorro realizado
                if (setqueryres==1){//si todavia esta en las ultimas 700/500/300 consultas
                        sprintf(out2, "./salidas/resquery%d", maq+1);
                        arch4 = fopen(out2, "r");
                        while(feof(arch4)==0){//imprime las consultas
                   fgets (buffer1, 80,arch4);
                   sscanf (buffer1,"%s %s %s %s %s %s %s
%s",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8,buffer9);
           //printf ("%s %s %s %s %s %s %s %s
\n",buffer2,buffer3,buffer4,buffer5,buffer6,buffer7,buffer8,buffer9);

          //si son iguales las consultas comparadas se realiza el ahorro
```

```
                    if (f_data[0]==atof(buffer2) && f_data[1]==atof(buffer3) &&
f_data[2]==atof(buffer4) && f_data[3]==atof(buffer5) &&
    i_data[0]==atoi(buffer6) && i_data[1]==atoi(buffer7) )
                            {
                                    ios=ios+atoi(buffer9);
                                    flagquery=1;
                                    records=atoi(buffer8);
                            }
                    }
                    fclose(arch4);
            }

            int just_retrieved_count=0;
            //printf("%f %f %f %f %f - %f \n", query_mbr[0], query_mbr[1],query_mbr[2],
query_mbr[3], query_mbr[4], query_mbr[5]);
            char *dup_block = new char [MAXBLOCK];
            for (int i = 0; i < OBJECTNUM; i++)
                    retrieved_record [i] = 0;

            int io_count_before_query = mvr_tree -> file -> iocount; //nodos accesados
despues de realizar la consulta

            if(flagquery==0){
            mvr_tree -> rangeQuery(query_mbr, result_list, retrieved_record, dup_block);
             just_retrieved_count = result_list -> get_num();
            }
            else{
             just_retrieved_count=records;
            }
            int io_count_after_query = mvr_tree -> file -> iocount;



            retrieve_count += just_retrieved_count;

            if(query_mbr[4]==query_mbr[5]) contTs++; //si los time-stamp son iguales,
consulta = time-slice
            else contTi++;//si son distintos, consulta = time-interval

            result_list -> print();
            //printf(" setqueryres = %d ; just_retrieved_count= %d
\n",setqueryres,just_retrieved_count);
```

```
            if (setqueryres==0 && just_retrieved_count>0){
            sprintf(out2, "./salidas/resquery%d", maq+1);
                        arch5 = fopen(out2,"a+");

            fprintf(arch5,"%f %f %f %f %d %d %d %d \n", query_mbr[0],
query_mbr[1],query_mbr[2], query_mbr[3],
(int)query_mbr[4],(int)query_mbr[5],just_retrieved_count,io_count_after_query -
io_count_before_query);
        fclose(arch5);
            }

            delete [] dup_block; delete
            [] query_mbr; delete []
            retrieved_record; delete
            result_list;

            query_count  ++;

            printf("\n Procesador\t %d \t Query \t %d \t: \t %d \t records retrieved in\t %d
\tI/O's ",maq+1, query_count,just_retrieved_count, io_count_after_query -
io_count_before_query);

    }

    int new_io_count = mvr_tree -> file -> get_iocount();
    printf("\n Total \t %d \t records retrieved in \t %d \t I/O's", retrieve_count, new_io_count
- old_io_count);
    //buffer=0;
    if (buffer==1){
      v2=0;
      do{
      v2 = rand() % 100 + 1;
      }while(v2<10 || v2>20);
      v2=v2/100;
      v2=1-v2;
            fprintf(arch,"Total: %d,procesador: %d descontado \n", (int)((new_io_count -
old_io_count)*v2),maq+1);
    }
            if (buffer==0){
            fprintf(arch,"Total: %d,procesador: %d \n", (new_io_count -
old_io_count),maq+1);
```

```
    }
     if(query_count >= 700){
                        sprintf(out, "./salidas/flag%d",maq+1);
                        arch2 = fopen(out,"w");
                if(contTi>contTs){
                fprintf(arch2,"0");
                }
                else{
                        fprintf(arch2,"1");
                }
                fclose(arch2);
     }

    fclose(query_file);
    delete [] f_data;
    delete [] i_data;

    fclose(arch);

    return new_io_count - old_io_count;
}
```